AD-A238 736

(1)

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GNE/ENP/91M-3

HARDWARE UPGRADE OF A SEGMENTED
DRUM ASSAY SYSTEM

THESIS

Claude A. Irvine, Captain, USAF

AFIT/GNE/ENP/91M-3

AFIT/GNE/ENP/91M-3

HARDWARE UPGRADE OF A SEGMENTED DRUM ASSAY SYSTEM

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Nuclear Engineering

Claude A. Irvine, B.S.

Captain, USAF

March 1991

Preface

This thesis describes the conversion of a drum assay measurement computer system (Digital Computer System PDP-11/05), at EG&G Mound Applied Technologies, with an IBM PC. The conversion of the computer system was primarily a team effort. For this, I am indebted to Mr. Allen Campbell and Steve Rowe of the Safeguards Applications Development and Nuclear Material Control Group at Mound, and to Ernie Tyra of the Electrical Engineering Group. Also, many thanks to Major Beller, my thesis advisor at the Engineering Physics Department, who made this all possible.

Claude A. Irvine

Table of Contents

Table of Figures

Table of Tables

Abstract

This paper describes the conversion of a DEC PDP-11/05 computer system, previously used in Canberra's Model 2220B segmented gamma scanner, with an IBM PC. Two tasks necessary for completion of the project involved reestablishing communications with a Canberra Series 35+ multi-channel analyzer and a scan table controller. An additional serial/parallel card was installed in the PC to reinstate communications with the multi-channel analyzer. For computer control of scan table controller operations a digital input/output card was used along with an external electromechanical relay board; when implemented together this hardware setup replaces functions that were normally processed through a motion control interface card housed within the DEC. Software consisted of Canberra's PC Toolkit while newer programs were written in Microsoft's QuickBASIC 4.5 and Macro Assembler 5.1. Five codes were written--two of these are device drivers written in assembly language and the other three are menu and control programs written in QuickBASIC. The modification enables simplified programmer enhancements.

HARDWARE UPGRADE OF A SEGMENTED DRUM ASSAY SYSTEM

I. *Introduction*

During the past several years EG&G Mound Applied Technologies has been using a Canberra Model 2220B segmented gamma scanner for measuring the amount of ^{238}Pu in waste drums. This system has worked well, however the computer, a DEC PDP-11/05, was difficult to program and implementing additional enhancements proved even more formidable. Thus a decision was made to replace the outmoded DEC with a small personal computer, an IBM PC.

However, the up-to-date computer system would still have to manage control over the existing equipment. Especially important was initiating operation of a Canberra Series 35+ multi-channel analyzer (MCA) and a scan table controller (STC). In the DEC system, a motion control interface (MCI) card controlled the STC by acting as a switch closure device; so as a substitute in the PC, a digital input/output card was used in conjunction with an external electromechanical relay board. For reconnection of bi-directional communication circuits between the PC and the MCA a serial/parallel card was installed. Finally software was developed to run the modified drum assay system.

Next, the development of the DEC to PC conversion is explained by subdividing the following report into two parts. For the reader unfamiliar with drum assay measurements, the first part is a short discussion of the equipment used and how a drum assay procedure is performed. The second part (Chapter 3) is a detailed discussion of the DEC to PC conversion. Most of the drum assay equipment is shown in Figures 1.1 and 1.2, with Figure 1.2 showing the present system at Mound as a result of the computer upgrade.

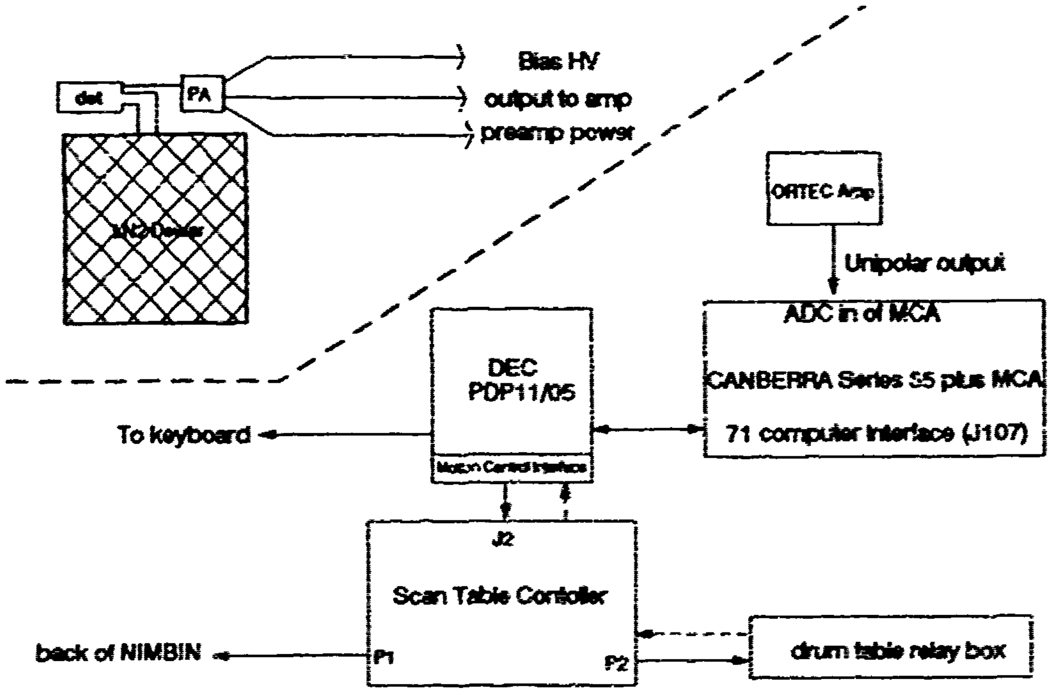

Fig. 1.1 Original Drum Assay System Configuration

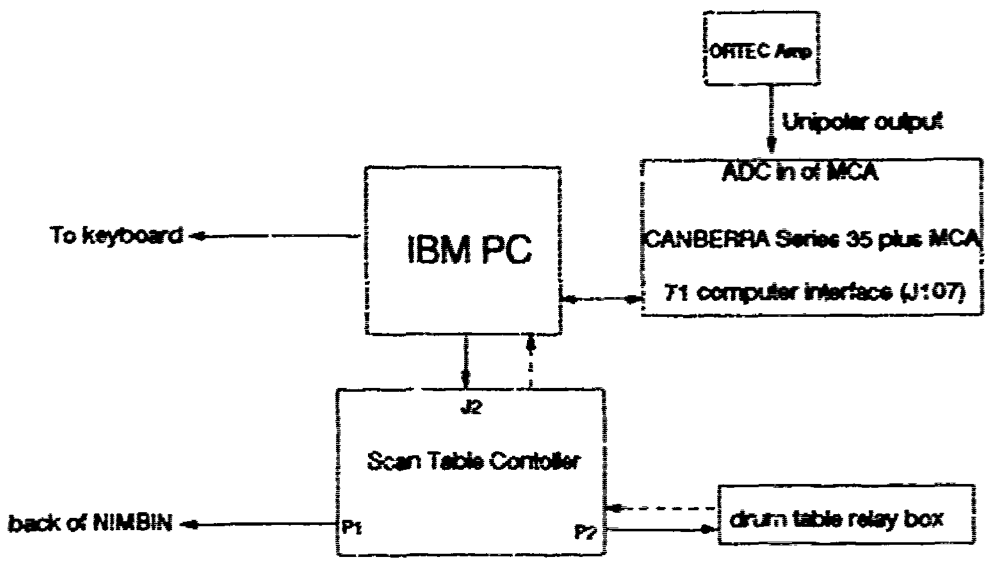

Fig. 1.2 Present Drum Assay System Configuration

II. Principal Methods of Drum Examination

The simplest, non-invasive method for assaying radionuclides in sealed drums is by spectroscopy of gamma rays that are sufficiently energetic and intense to penetrate the walls of the drums. In most instances, spectra obtained by a high-resolution detector, such as an intrinsic germanium coaxial detector used at Mound, will be characteristic of specific radionuclides. However, the success of an assay depends on the activity of the source, the mode of decay of the radionuclides and whether they emit gamma rays of sufficient energy to penetrate the walls of the drum. Complications are further introduced by the shielding factor of the drums and drum contents. Yet in most cases the determination of the radionuclide identities are readily obtained by examination of the gamma peaks in a spectrum.

Drum Assay Measuring Equipment

A complete drum assay system usually consists of a number of distinct, separately-bought nuclear instrumentation modules, or instead, a complete system can be purchased. There are several companies that build integrated drum assay measurement systems, only one in particular is discussed in this report, the system used by Mound which is manufactured by Canberra Industries, Inc.

The following is a list of equipment of the drum assay measurement system:

1. A gamma-ray detector, to determine the energy of the absorbed gamma rays.

2. Pre-amplifier and spectroscopy amplifier, for amplifying and shaping the signal.

3. Multi-channel analyzer, for determining the gamma-ray peaks.

4. Computer, for analyzing the data.

5. A drum table platform, for manipulating waste barrels.

The above listing does not include ancillary items such as power supplies, drivers, cables, etc., nor does it account for software that is critical for controlling an entire system.

Drum Assay Procedure

To begin the procedure a sealed drum of known radioactive waste, called the calibration standard, is placed on a rotating platform and elevated to expose its lower side to a collimated, gamma-ray detector. Next, the drum is scanned by the detector for a preset duration. Then the drum is lowered to another position and another scan is completed. Note: each successive step of the moving stage is controlled through the use of computer software. Finally, once the entire length of the drum has been scanned, the computer generates a report of the contents based on the radionuclides identified from the gamma-ray spectra observed (see Appendix A for a sample print-out).

One purpose of the calibration standard is to determine the absolute efficiency of the detection system. This information is then used as input data for future drum assay measurements. In addition, the data serves as a benchmark for subsequent measurement control runs. For this purpose the *Mound Technical Manual*[1] specifies that calibration data be taken both before and after each session of drum assays to confirm that the detection efficiency of the system hasn't changed. After completing a calibration run, successive assays following the same routine can be repeated for one or more drums.

III. Hardware Upgrade at Mound

The scanning system modified at Mound Nuclear Safeguards and Development Group was a Canberra Model 2220B. Since the purpose of replacing the DEC with a PC was to attain a more efficient and reliable scanning system, a spin-off would be simplification of code enhancements. However, modernizing the hardware necessitated installation of three boards--a serial/parallel card, a parallel input/output card and an electromechanical relay board--to replace functions formerly controlled by the DEC. This chapter discusses installation of those items and of software developed to enable PC-to-STC hardware handshaking and to link Canberra-supplied communication programs to the MCA.

Microsoft QuickBASIC was the primary language for software development. Hence all software programs would run inside, or in conjunction with, the QuickBASIC Interpreter. Hardware drivers were written in Microsoft MACRO and then compiled and linked as a QuickBASIC library. As a consequence, the library module has to be loaded when QuickBASIC is started.

Problem Definition

To complete the changeover, the PC needed to interface with existing equipment by performing three main functions:

1. Communicate bi-directional data transfers with an external MCA, via serial interface for command/control functions and parallel interface for data transfers.

2. Function as the control panel by supplying driver signals and sensing feedback signals to and from the STC (the STC drives trolley motors and lifts the drum turntable up and down, and performs other various functions).

3. Analyze spectral data taken by the MCA.

Furthermore, development of new software would be needed to replace existing codes. Originally the DEC PDP-11/05 controlled input and output functions by using BASIC/RT-11 language programs provided by Canberra. This software consisted of 3 main modules:

1. A BASIC language interpreter, with callable machine language functions for system control and data acquisition.

2. A calibration program.

3. A general purpose program that included several BASIC subroutines for allowing the computer to control MCA functions, like setting up a region-of-interest (ROI), doing a spectral peak search, or presetting counting time, etc. (a more universal set of subroutines that parcels control and data transfers between the DEC and MCA is given by Larry V. East[2]).

Because all of the other drum assay equipment was retained, newly written programs for the PC would have to execute identical tasks to the above BASIC/RT-11 language programs. Thus the modified drum assay system would still operate similarly to that of the original DEC system.

Electrical Mapping of Hardware

This section discusses installation of new hardware necessary to connect the PC with the MCA and STC. MCA communication hardware consisted of a parallel driver card

(commonly called the Fast IBM Interface or 3576) and a serial communication card (just referred to as the PC Interface or 3575). PC to MCA electrical connections were established by installing a serial/parallel (S/P) adapter board in slot #2 of the PC, and coupling both serial ports by way of a 9-pin null modem cable, 9-pin to 25-pin adapter, and 25-pin ribbon cable. Similarly, the parallel ports were connected with a shielded 25-pin cable. Figure 3.1 shows the wiring diagram and cable connections.

Additionally, since there was a limited number of serial ports on the PC, a Clear Signal T-Switch Box was spliced into the serial line so other applications could utilize the same serial port. Serial hardware handshaking is enabled when the switch on the box is in the "A" position.

The baud rate for the logic board in the MCA was set to 1200bps. Higher transmission speeds are also possible, however, the *CANBERRA PC Toolkit Software Manual*[3] advises if Microsoft Windows is to be used the baud rate of the serial port should be less than or equal to 1200bps, thus assuring no characters are lost in the transfer process. It was decided to limit data transmission to 1200bps for now, thinking that in the future windowing capability might be applied.

Because a common setting of 1200bps on both PC-to-MCA communication boards was used, installing both serial and parallel communications were a necessity because of the huge

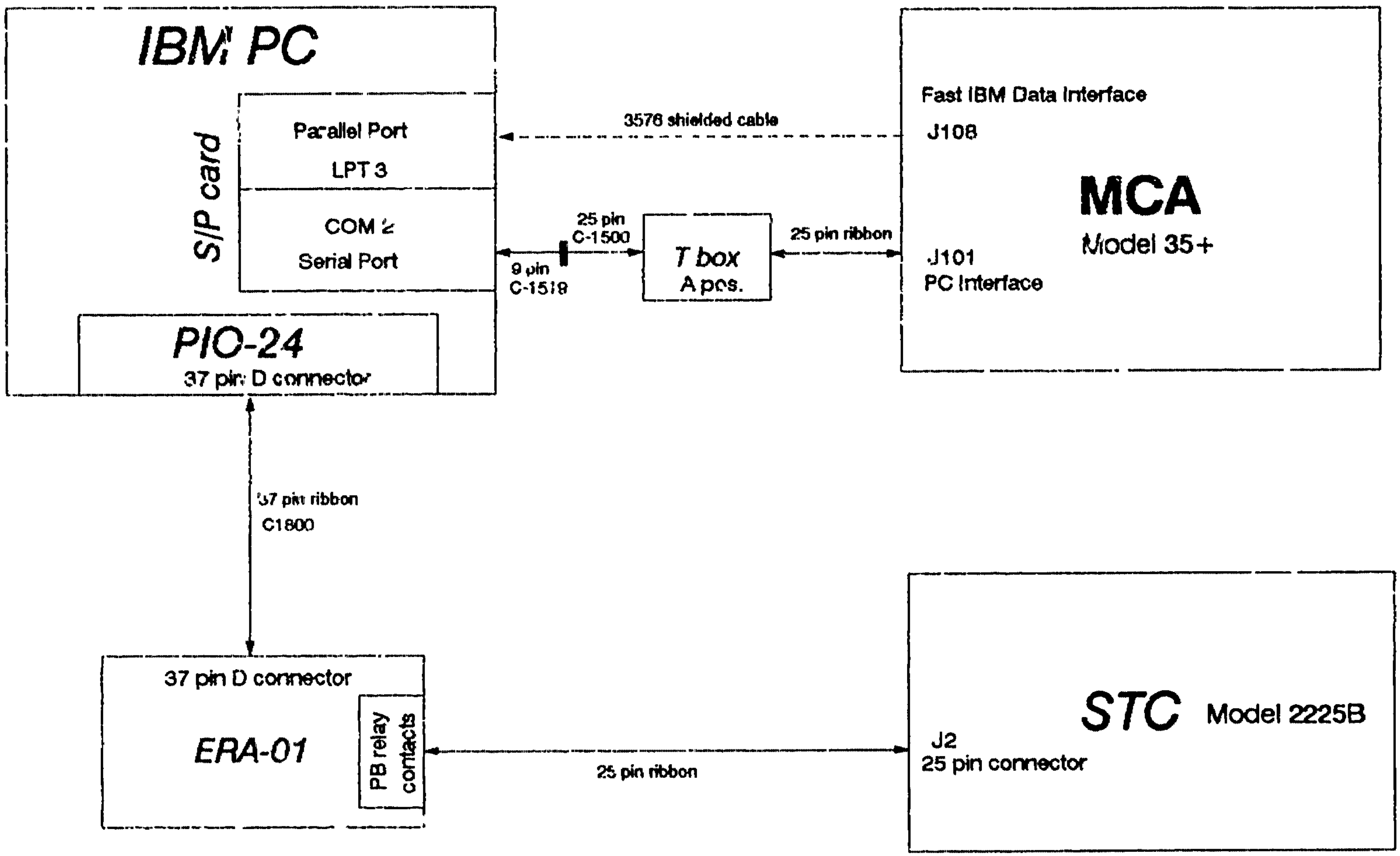

Figure 3.1 PC to MCA/STC Wiring Diagram

amount of data sent over the lines (the MCA could send up to 8192 channels of information). Transferring that amount of data through a serial line would take up too much time, so to minimize the transmission interval, command and control functions were routed over a serial line while the [illegible] transfer used parallel exchange. Note however, parallel [illegible] transfer was not critical since all MCA functions could be handled through serial communications.

For PC-to-STC communications, a high-output current, parallel digital interface (PIO-24) was installed in PC slot #5 (base address 304 hex), along with an external electromechanical 8 channel SPDT relay board (ERA-01), to route PC to STC communications. All digital input/output lines from the PIO-24 are connected to the ERA-01 via a MetraByte C1800 ribbon cable. Cabling from the ERA-01 to STC is by way of a 25-pin ribbon cable. Again, Figure 3.1 shows the wiring diagram for cable and equipment connections, and Figure 3.2 shows a schematic of the electrical connections.

The PIO-24 and ERA-01 devices are the functional equivalent of the MCI card (recall Figure 1.1). Originally, the MCI served 7 control lines by sinking 300 ma to common ground of an EG&G ORTEC NIM power supply. But the PIO-24 could only sink 64 ma to ground, so it was used to drive the ERA-01. Sink current is maintained for 7 control lines through a set of relay contacts on the ERA-01 board. In addition, 5 sense lines from the STC enter the ERA-01.

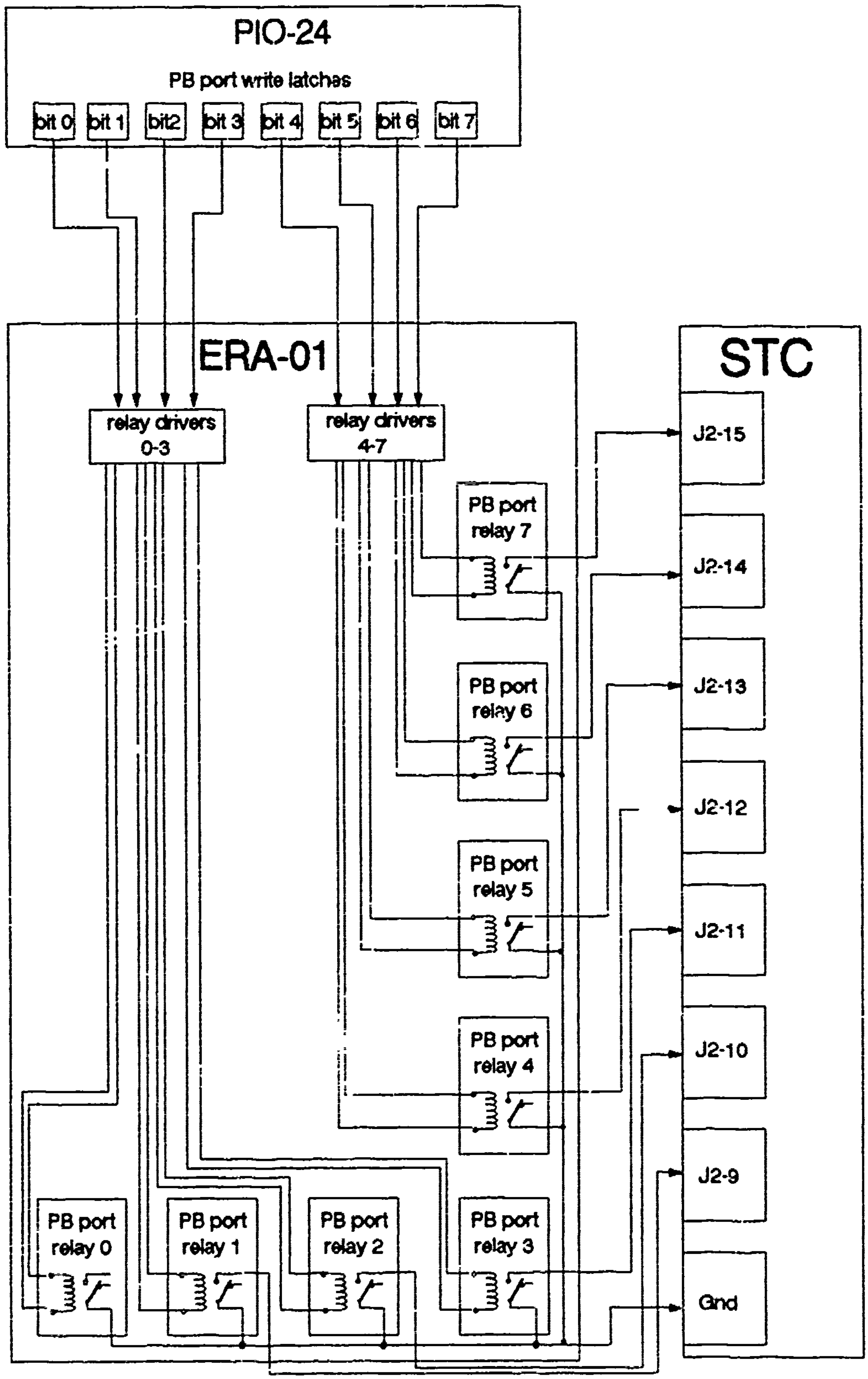

Figure 3.2 PIO-24 to STC Schematic

These lines are for sensing contact closure only and are presently disconnected. For more information on control and sense lines, see the next section, "Establishing PC to STC Communications."

Three of the 7 PIO-24 control lines drive 3 relays on the STC: K1, K2, and K3. Output from K1 and K2 set the "return" and "advance" contacts of 6 small relays on the drum turntable panel box. The panel box has 7 large contact relays (CR's 1-7) which drive the left and right trolley motors and drum rotation motor; the small relays (CR's 8-13) are the relay drivers for CR's 1-7. Output from K3 opens and closes a shutter used by the transmission source. The other 4 control lines (1-4) turn on display lamps on the STC panel.

Appendix J shows each control line connection on the PIO-24, ERA-01 and STC. For example, control line 5 is PB port bit 5 on the PIO-24, PB port relay normally open contacts on the ERA-01, and pin connection J2-13 on the STC. In actuality a port bit does not directly join end to end a port with its relay contacts, however, what is meant is the port bit can energize a corresponding relay on the ERA-01. Furthermore, the "common" contacts of all the relays are connected to J2-25, which is common ground of the NIM power supply. Since the contacts of the PB port relays are normally open, no connection is made to ground. However, setting a PB port bit high on the PIO-24 (through TableSYS.ASM)

will close the proper relay on the ERA-01 and short the normally open contacts to ground--in essence, activating a control line. In this way the PIO-24 and ERA-01 combination can replace functions formerly managed by the MCI.

Establishing PC to MCA Software Communications

The Canberra S370 PC Toolkit is a collection of driver and utility programs (17 total) useful for setting remote control communications between a Canberra MCA and an IBM compatible computer. The 3 most important programs are PCUTIL.BAS, MCAS.COM and SETMCA.COM (page 13 of the *PC TOOLKIT SOFTWARE* manual provides an in-depth explanation of each program). These programs establish serial communication. Both MCAS and SETMCA are executable device drivers while PCUTIL is a GWBASIC utility program. The essential codes for parallel communication are BINS.COM, BINSTUFF.OBJ and FASTRAN.EXE. Both FASTRAN and BINSTUFF are FORTRAN test programs, while BINS is the parallel device driver. Since all 6 programs were vital in governing MCA operations, each one was installed.

The first prerequisite was to check the settings of the S/P board port addresses. By default, initial factory settings activate software handshaking using serial port #2 and parallel port #3. Port allocation was verified by CHECKCOM.COM, a program that verifies port availability and determines whether or not the port is suitable for serial or

parallel communications. Then the device drivers, MCAS.COM and BINS.COM, were installed in CONFIG.SYS. Appendix B lists a print-out of CONFIG.SYS. Since the jumper in the S/P card was set to COM2 and LPT3, the device driver arguments in CONFIG.SYS (for MCAS.COM and BINS.COM) are COM2 and LPT3, respectively.

Finally, SETMCA was added to AUTOEXEC.BAT. Appendix C lists a printout of AUTOEXEC.BAT. SETMCA sets the software for serial communication to operate at 1200bps--the same as the baud rate setting on the MCA logic board. Note: when changing the baud rate, failure to set the logic board and SETMCA to the same value will disable serial handshaking.

Since Canberra specified serial communications could be accomplished in either BASIC or FORTRAN languages, and parallel data transfer by using FORTRAN only, a principal language needed to be selected. The Microsoft QuickBASIC Interpreter became the primary language for code development because of its outstanding debugging capabilities. However, this meant FASTRAN had to be rewritten in QuickBASIC with BINSTUFF converted over to a QuickBASIC library, and lastly, PCUTIL needed to run inside the interpreter.

Proper operation of the serial communication line was checked by running PCUTIL under GWBASIC. Several commands were given, e.g., initialize the MCA and set regions of interest, in verifying serial hardware setup. No problems were encountered.

DrumSYS.ASM. The original BINSTUFF was a collection of 5 FORTRAN subroutines that passed data back and forth to the device driver, BINS. However, BINSTUFF didn't work, at least in the QuickBASIC environment. Replacing the object code meant BINSTUFF had to be rewritten in assembly language, and also had to be Microsoft compatible in order to be linked as a QuickBASIC library. Appendix D lists the assembly source code for DrumSYS.ASM--essentially the same 5 procedures as in BINSTUFF but condensed into one code and data segment and rewritten with Microsoft Macro Assembler[4] directives/instructions and BASIC calling/naming conventions.

Two of the five procedures are documented in-depth, while the other three share instructions similar to those observed in the first two procedures. And all of them foster DOS interrupt functions and standard File Control Blocks.

Because documentation on BINS and BINSTUFF was nonexistent, it made deciphering the original BINSTUFF difficult. Apparently DEVICE is used as a symbol for the ASCIIZ string, 'BIN1.' This means BIN1 is a device (opened by BINS) for writing data to during a parallel data transfer. In the same fashion, FILEHANDLE is a symbol that contains the file handle.

DOS interrupts open and close the file BIN1. For example, the first interrupt (INT 21/3D) in the OPENBI procedure accesses the file private to the current process. On a successful return, BIN1 is opened with the read/write pointer at the beginning of the file. The file handle (a 2-byte number, or word) is used in later DOS interrupts to reference back to BIN1. Actually, file handle is the 2-byte offset address while DS (Data Segment register) points to the segment address, also 2 bytes.

After BIN1 is opened, the next procedure, INBIN, performs the actual data transfer. Now parameters are passed to the procedure along with the return address. A requirement of BASIC is to declare assembly-language procedures as FAR. Hence, the return address is 4 bytes long. This may not seem important at first, but when DrumSYS.BAS calls these procedures it makes a difference. The reason for this is the way BASIC calls an assembly-language program. By default, BASIC passes parameters to a procedure by reference as a 2-byte address[5]. But as will be seen later on in DrumSYS.BAS, the parameters are passed as LONG, i.e., a 2-byte segment address in both SS and DS registers plus another 2-byte offset address--4 bytes total. First, BP, which serves as a framepointer, is pushed onto the stack. Next BP is loaded with the last value SP (Stack Pointer) pointed to--the old value of BP. Since BP never changes in the procedure, all 4-byte parameters pushed onto the stack,

and the 4-byte return address, can be referenced relative to BP. Thus, if 2 parameters were pushed onto the stack, [BP+12] would point to the first parameter, [BP+8] would point to the second parameter, and finally, [BP+4] would point to the return address. Then it is a simple matter to access the data as required in the rest of the procedure's body.

In summary, the original procedures from BINSTUFF were rewritten, and the new assembly-language program is called DrumSYS.ASM. Next, DrumSYS.ASM was made into a QuickBASIC library. From the DOS prompt, DrumSYS.ASM is compiled into an object code using the command:

> MASM DrumSYS.ASM /zi

The option /zi is an assemble-time option that produces an object file in CodeView format. CodeView is a symbolic debugger from Microsoft and is useful for troubleshooting machine code. For now, the compiled code called DrumSYS.OBJ is linked as a QuickBASIC library using the command:

> Link /q DrumSYS.OBJ,DrumSYS.QLB,,BQLB45.LIB /co

Again, the /co option is used for CodeView only.

To call the procedures inside the QuickBASIC Interpreter, QuickBASIC has to be loaded with the command:

QB /l DrumSYS.QLB

Now the QuickBASIC Interpreter has all five procedures available for accomplishing parallel data transfers. In the

following section, the conversion of FASTRAN into the code DrumSYS.BAS is discussed along with how parameters have to be defined before they can be passed to DrumSYS.QLB.

DrumSYS.BAS. As pointed out earlier, BINSTUFF was to be called from a FORTRAN program called FASTRAN. FASTRAN was a test program that could transfer data files (MCA spectra) on disk to and from the MCA. It used a serial line for sending simple command functions while data were dispatched over a parallel line. The new FASTRAN, rewritten in QuickBASIC, is called DrumSYS.BAS. Appendix E lists the QuickBASIC source code for DrumSYS.BAS--practically the same lines as in FASTRAN but rewritten with Microsoft QuickBASIC subroutines and BASIC calling/naming conventions.

Another addition to DrumSYS.BAS was the ability to chain to Canberra's utility program PCUTIL.BAS. However, PCUTIL.BAS would not run in the QuickBASIC Interpreter, so it was modified. Because of the length of this program (approximately 800 lines of code), it is not listed here. The new program (PCUTILQB.BAS) is the same as Canberra's except with a few minor changes.

DrumSYS.BAS is adequately documented, so only a short discussion is included here. However, there are two important peculiarities. One has to do when DrumSYS.BAS is first started, and the other when parameters are passed to DrumSYS.ASM.

The serial device driver MCAS.COM opens two devices, denoted MCAIN and MCAOUT. These devices are used for MCA communication; note: they are not "ordinary" files. DrumSYS.BAS will write information to MCAOUT and read data from MCAIN in order to communicate with the MCA. If the MCA does not respond and an error condition occurs when DrumSYS.BAS opens MCAIN for input, press the INDEX and HOME keys (on the MCA front panel) simultaneously to clear the MCA and enable handshaking. In the event the above does not work, then press the YES key (on the MCA front panel) and retry. The previous steps should only be needed when the MCA is first turned on. The reason is because on the back of the MCA there is a switch that has three positions: Remote, Shared, and Local. When the switch is in the Shared position both the front panel keyboard and a remote computer can enable handshaking. However, during a cold start-up and when the switch is in the Shared position, the MCA electrically pulls one of the serial line pins low, thus disabling remote communications. The above procedure reinstates remote communications.

In DrumSYS.BAS parameters are passed to DrumSYS.ASM as LONG, i.e., a 2-byte segment address plus another 2-byte offset address. This implies numerical data must be declared LONG (4 bytes). In QuickBASIC an integer data type is only 2 bytes long, so failing to pass values as 4 bytes will eventually cause the computer to hang. The problem is

solved in DrumSYS.BAS with the "DIM SHARED IAR(8194) AS LONG" statement. Another concern is how the array IAR is passed. BASIC uses an "array descriptor" to pass arrays. The array descriptor lets BASIC access an array by pointing to the first element of the array's address. This is necessary because BASIC allocates computer memory dynamically; thus an array may shift in memory location when the program is run. Therefore QuickBASIC VARPTR and VARSEG functions must be used when passing an array. However, one exception to this rule is if the array elements are passed by value. For more information on BASIC array descriptors see Microsoft's *Mixed-Language Programming Guide*.

Establishing PC to STC Software Communications

The STC (model #2225B) is the rack mounted control panel for the gamma scanner[6]. It is the main panel for starting an assay measurement sequence. The panel performs four central functions: SYSTEM POWER, SEGMENT SIZE CONTROL, TABLE ROTATION and PROGRAM CONTROL. These functions are summarized in the *Segmented Gamma Scanner Operating Manual* as follows:

1. SYSTEM POWER: Turns on the AC power to the drum rotator panel and contains the emergency stop button.

2. SEGMENT SIZE CONTROL: Limits the length of drum segments. Use of the RETURN button moves the drum table to the top position and using the ADVANCE button indexes timer to advance the drum table downwards. Note: RETURN and ADVANCE are computer controlled relays.

3. TABLE ROTATION: ON position the trollies forward and starts drum table rotation. OFF position moves the trollies backward and stops drum table rotation.

4. PROGRAM CONTROL: SHUTTER opens and closes the shutter. ABORT terminates the drum assay measurement. CALIB initiates the calibration procedure. INIT allows for manual calibration of drum. ASSAY starts the assay measurement procedure.

Two of the functions, SEGMENT SIZE CONTROL and PROGRAM CONTROL, deploy "digital" input/output lines to implement a drum assay procedure.

Presently there are 12 digital lines, 7 control lines and 5 sense lines. These are summarized in Table 3.1. The PC invokes operation of the control lines through TableSYS.ASM, the software driver for the STC. Right now all 5 sense lines are disconnected at the ERA-01 because of earlier troubles that were encountered when AC cross talk would feed back into the parallel digital interface and destroy integrated circuits; nonetheless, their functions have been replaced by using a time-delay procedure (in TableSYS.ASM) called Pause(x).

Most of the mechanics of the drum assay utilize control lines 5, 6 and 7. For example, invoking control line 6 (relay K2 energized) will activate a drum segment size timer; the timer is preset by rotating several thumb-wheel switches on its front face to a desired setting. Once the timer energizes it lowers the drum table platform until time has run out. Thus, the length of the segment drop is lim-

ited by the setting on the timer and remains constant throughout the drum assay procedure. Program flow will then issue commands to the MCA through CALIB.BAS (see Appendix H) to start collecting data. Issuing another control 6 again lowers the drum, and this cycle repeats until all segments have been analyzed. Similarly, control 7 (relay K1 energized) moves the drum table platform upwards, and control 5 (relay K3 energized) opens and closes the transmission source shutter.

TABLE 3.1
STC Control and Sense Lines

Control Line	Purpose
1	Turn ABORT lamp on.
2	Turn CALIB lamp on.
3	Turn INIT lamp on.
4	Turn ASSAY lamp on.
5	Open transmission source shutter and turn SHUTTER lamp on.
6	Move drum table downwards.
7	Move drum table upwards.
Sense Line	
1	Signals ABORT button has been pressed.
2	Signals CALIB button has been pressed.
3	Signals INIT button has been pressed.
4	Signals ASSAY button has been pressed.
5	Signals drum table is advancing.

TableSYS.ASM. TableSYS.ASM is the software driver for the PIO-24. It consists of 4 public assembly-language procedures (public means the procedures are accessible from another high-level language). These are

(1) SetCRconfig(): initializes PIO-24 mode, no parameters are passed;

(2) Set(x): will energize relays 1-7, pass numbers 1-7;

(3) Clr(x): will turn off relays 1-7, pass numbers 1-7;

(4) and Pause(x): will suspend program execution for 1 to 59 seconds. Procedure uses DOS system time and is independent of processor speed, pass numbers 1-59.

Appendix F lists the TableSYS.ASM source code, and Figure 3.3 shows a flow chart of each procedure. More important, however, is every directive and instruction in the code has been commented should the PIO-24 port addresses need to be changed.

The "twenty-four" in PIO-24 stands for 24 digital input/output lines. These 24 lines are divided equally among 3 ports: PA, PB, and PC. All ports are configured input only when the computer is first turned on. However, when DrumTAB.BAS (see next section) is started it initializes the control register on the PIO-24 using the SetCRconfig() procedure. SetCRconfig() zeroes the contents of the write latches (all ports) and configures PB as a write only port. Currently the PA and PC ports are not used.

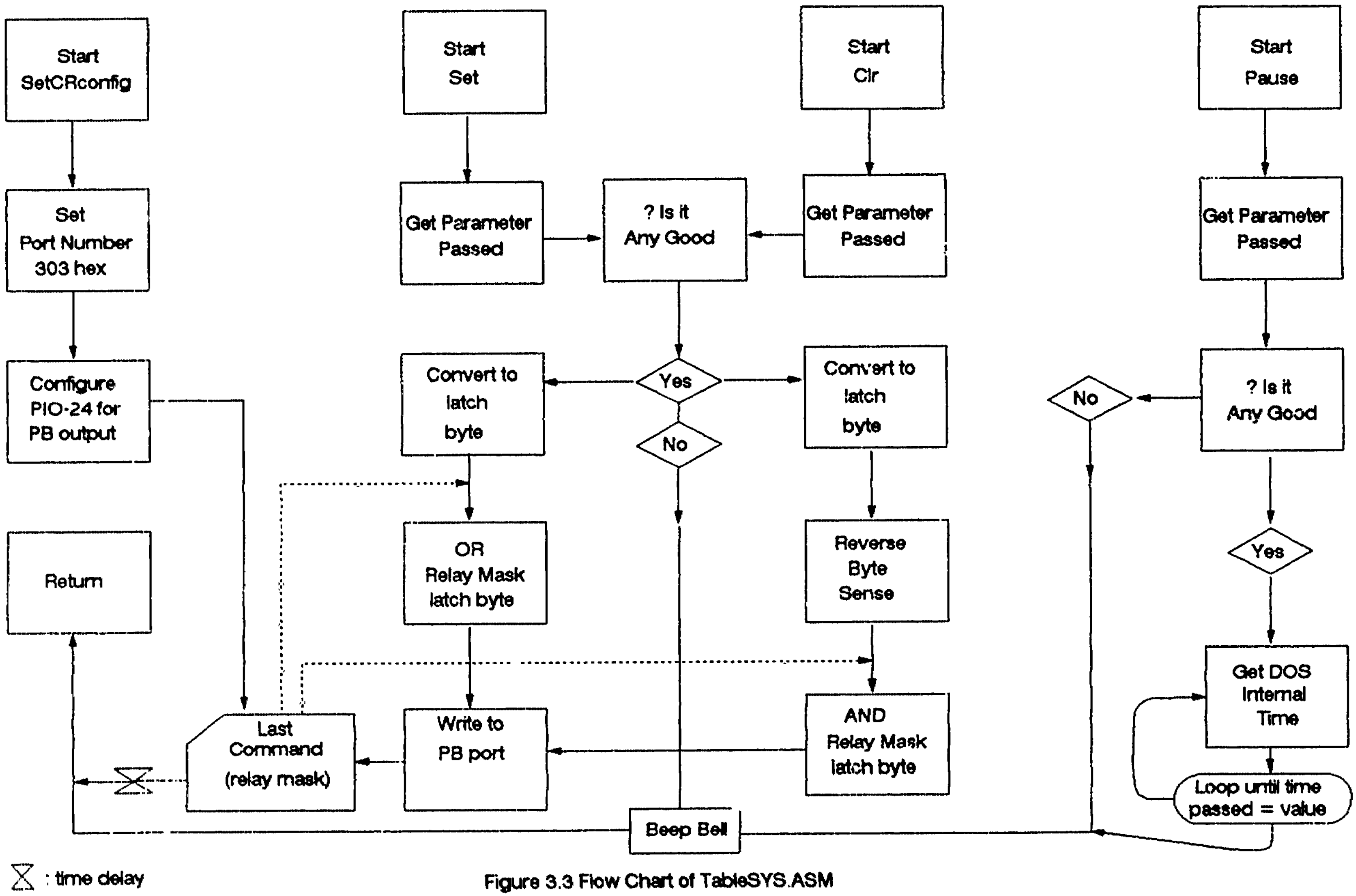

Figure 3.3 Flow Chart of TableSYS.ASM

Functionally, procedures Set(x) or Clr(x) drive the PIO-24 by retrieving a 2-byte integer parameter placed onto the stack segment. The variable x is the parameter passed, and ranges from one to seven. Accessing the value is the same as described in the *DrumSYS.ASM* section. The parameter has a one-to-one correspondence with control lines 1 through 7. For instance, a value = 3 means control line 3. Next, the parameter is encoded to the relevant PB port relay. This is accomplished by raising the number 2 to the parameter power. For example, a parameter equal to 3 is changed to $2^3 = 8$. The new value is then converted to an 8-bit number that corresponds to the PB port relay on the ERA-01 (see Appendix J). Since there are 8 PB port relays (0-7) and only 7 control lines, each control line is mapped (again a one-to-one correspondence) to one relay. Hence a parameter = 3 means control line 3, PB port latch bit 3, and PB port relay 3, with the byte number sent to the ERA-01 equal to 00001000 (in binary).

That's the way the process should work, however, there's one slight problem. When examining a drum it's necessary to set more than one control line. Changing the sense of one bit would turnoff the other relays (bit sense = 0), so to solve the problem a copy of the last command sent to the PIO-24 is saved in the data segment. The symbol LAST is a direct memory operand that represents the address (segment and offset) of the last command byte, and is referred to as

a "relay mask". In general, the relay mask is formatted as two 4-bit nibbles. The low-order nibble (bits 0 through 3) corresponds to control lines 1-3. Likewise, the high-order nibble (bits 4 through 7) corresponds to control lines 4-7. In a Set(x) procedure a bitwise logical OR on the PB port latch register (BL register) and relay mask is performed. The new byte is placed in the AL register and sent out to the PB port. As a result, only the relay that needs to be activated is turned on, or in the case of a Clr(x) procedure, the relay is turned off. Clr(x) works similar to Set(x) except it uses a bitwise logical AND on the port latch register and the relay mask.

Finally, the procedure Pause(x) is just a time delay with x being the number of seconds (between 1 and 59) to delay by. Since Pause(x) uses DOS interrupts, timing is independent of processor type. Hence an AT style computer would have the same delay as a 386 type machine. Mainly Pause(x) serves as a substitute for the sense lines that were previously used in the old RT-11 BASIC codes. The purpose of the sense lines were to let the computer codes know the STC was busy running the drum table platform. As an alternative, the programmer counts the number of seconds for the STC to perform a specific task (for example, moving the drum down one segment might take 5 seconds) and inserts

the appropriate delay in his/her codes using Pause(x). At first it might seem this appears like a cheap fix, however, it works well because of the constant RPM AC motors.

Like DrumSYS.ASM, compiling TableSYS.ASM into an object code is the same. In addition, the two object codes can be linked together to form one QuickBASIC library; then all of the procedures in both codes are available to DrumTAB.BAS, the menu program. Aside, an advantage of using assembly-language modules is that they can be interfaced from any high-level language without requiring modification should DrumTAB.BAS be written in another language.

The Central Menu Program, DrumTAB.BAS

DrumTAB.BAS is the main program that exploits all the features that have been talked about earlier. It can be started in the QuickBASIC Interpreter or compiled and linked separately into an executable code, DrumTAB.EXE, using Make-Drum (see next section).

Program flow starts, as seen in Figures 3.4 and 3.5, by setting the computer environment and initializing communications with the MCA and STC. Once successful, the code prints a menu on the display screen and then prompts the operator for his/her input. Depending upon which case was selected, the next step executes appropriate subroutines to perform the desired task. In the particular case of a drum assay procedure, DrumTAB executes Set(x) and/or Clr(x) pro-

cedures until it reaches the subroutine: Start Drum Assay Procedure. Then program management is transferred to an include file that controls MCA operations, like collecting ROI data, setting counting time, etc. Other than the exception of the include file, all operations are performed by DrumTAB--in conjunction with its library routines.

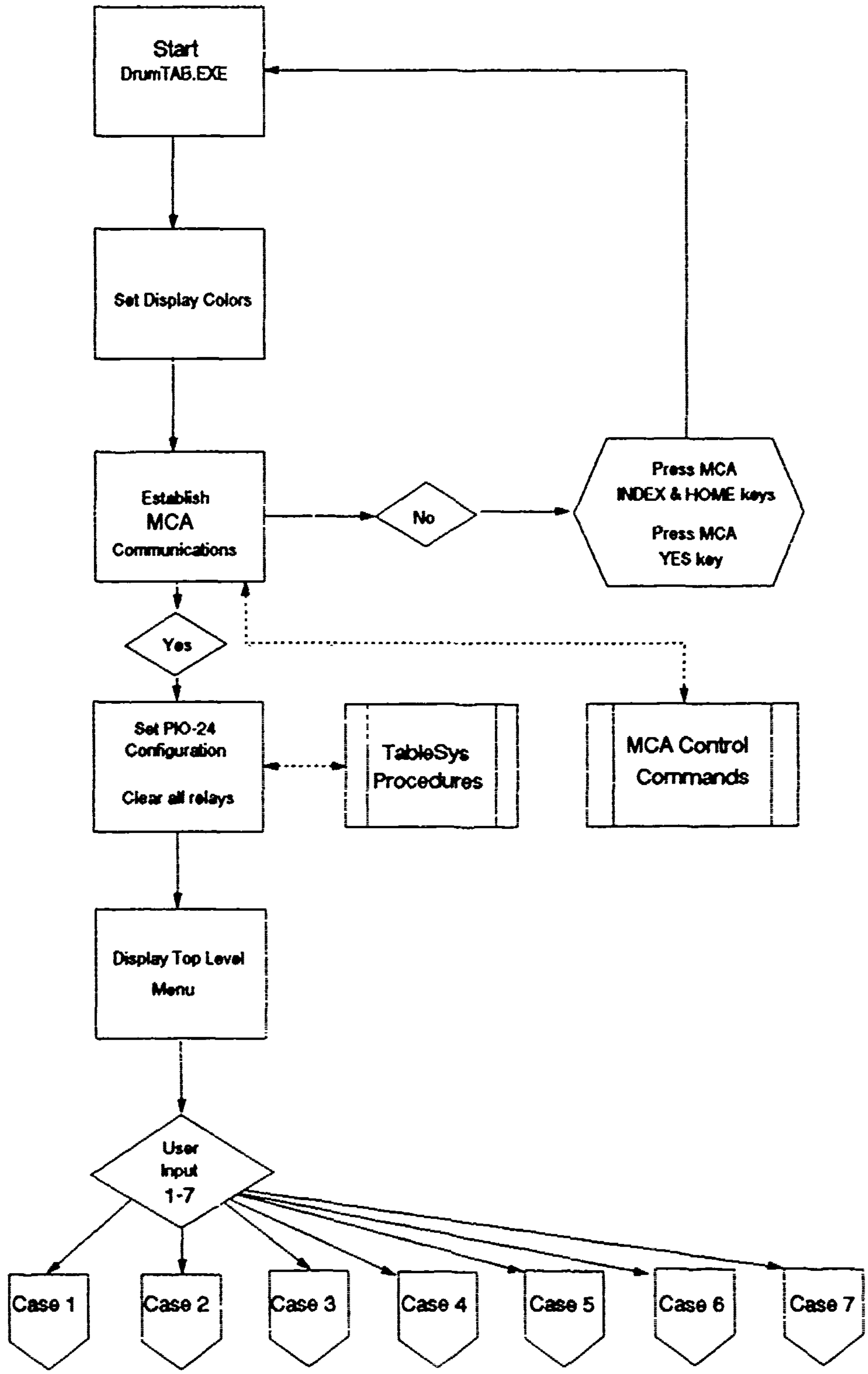

Figure 3.4 Flow Chart of DrumTAB.EXE

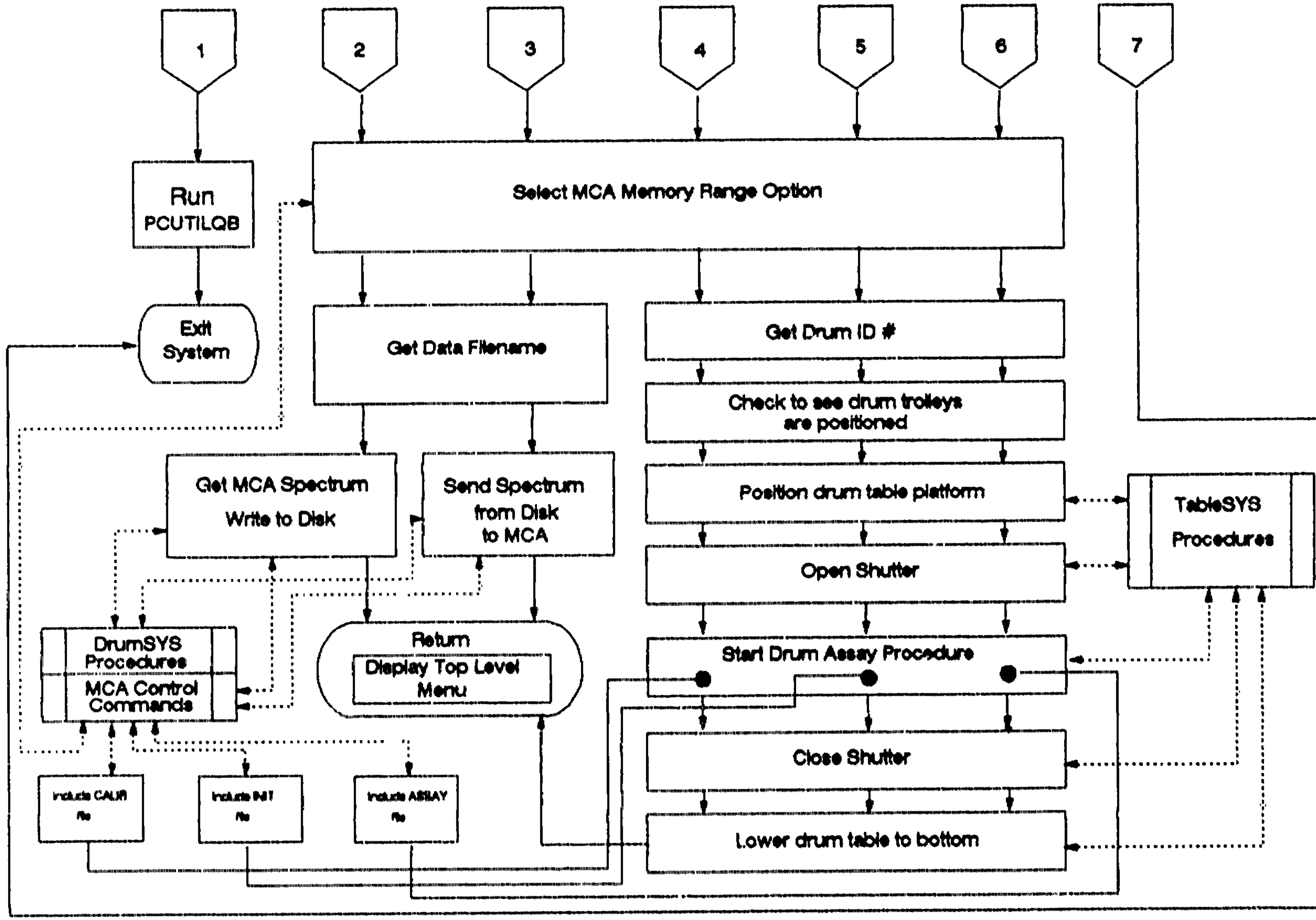

Figure 3.4 Continued

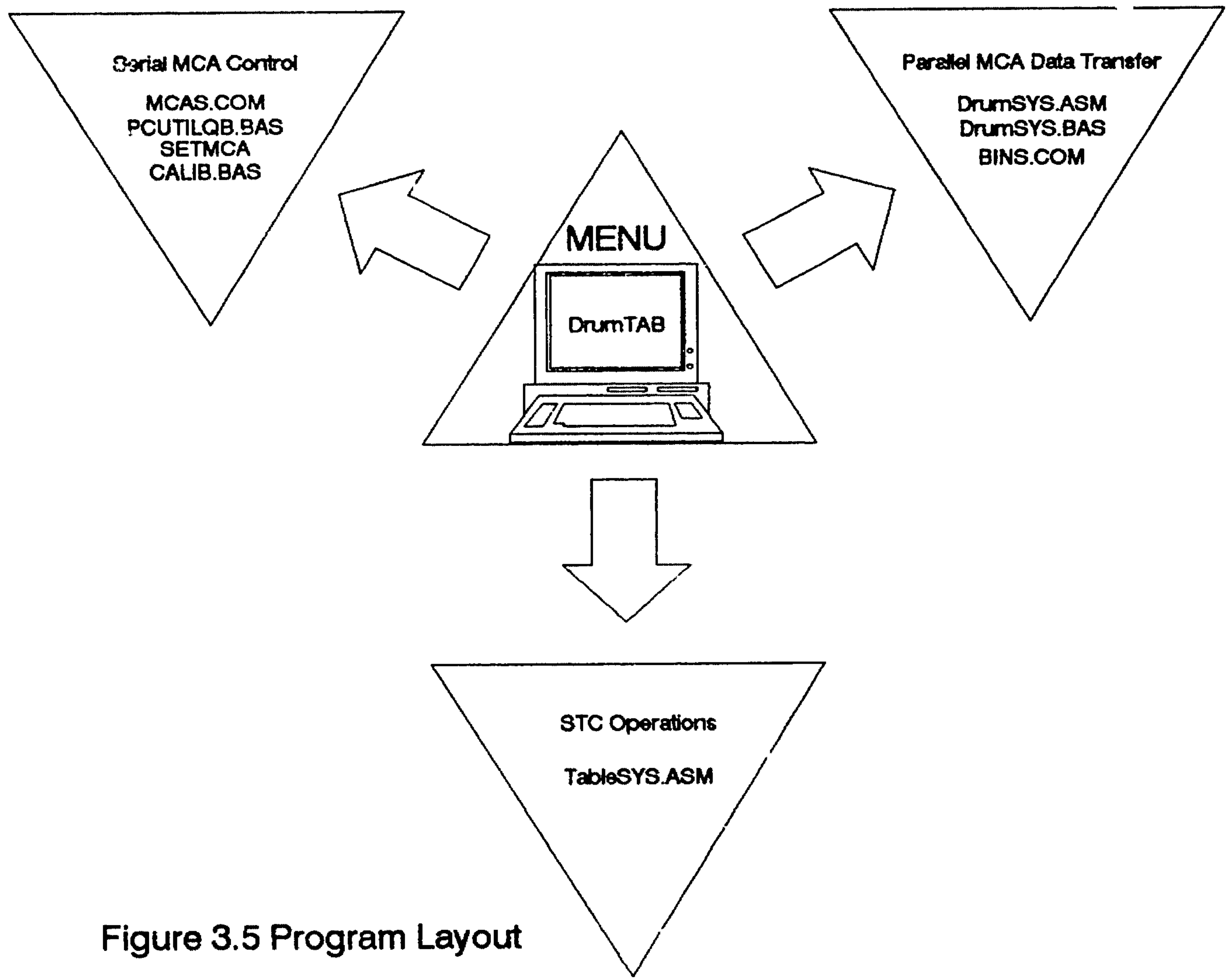

Figure 3.5 Program Layout

Compiling and Linking Codes Using MakeDRUM

To simplify compiling and linking all codes necessary to run DrumTAB.BAS outside the QuickBASIC Interpreter, a short code called MakeDrum is used to construct DrumTAB.EXE. Appendix I lists MakeDrum. It implements Microsoft's Make Utility to re-compile and link programs that have been modified since the last time they were compiled. Hence MakeDrum shortens the amount of time a programmer spends at the keyboard writing assembler commands, and makes changes to any of the programs a cinch to do.

IV. Future Efforts

The test file, CALIB.BAS, was used as an include file in DrumTAB to check whether or not a drum assay procedure could be accomplished with the new hardware setup. A sample output from CALIB.BAS is seen in Appendix K. This is similar to the DEC print-out in Appendix A. However, extra work is needed to correct the number of counts under the full-energy peak for absorption/scattering losses; also, no error analysis was performed. Hence two undertakings, an error analysis and addition of correction factors, must be considered part of a drum assay measurement. Included in Appendix L are mathematical derivations for the correction factors. Also, enhancements in the way of mouse support, a split screen graphics menu, and modular code design should eventually be incorporated. Nevertheless, these, and the efforts mentioned above, are left as suggestions for yet another project.

V. Summary

A DEC PDP-11/05 computer system, used in Canberra's Model 2220B segmented gamma scanner, was replaced with an IBM PC. Additional hardware for the computer upgrade included installation of a serial/parallel board, a digital input/output board (PIO-24) and an electromechanical relay board (ERA-01). Five computer codes were written: Table-SYS.ASM, DrumSYS.ASM, DrumSYS.BAS, DrumTAB.EXE and CAL-IB.BAS. TableSYS.ASM is a software device driver for the PIO-24. DrumSYS.ASM and DrumSYS.BAS are used with Canberra-supplied software for parallel data transfers. DrumTAB.EXE is a main menu and control program, and CAL-IB.BAS is an include file that interfaces with a Canberra Series 35+ multi-channel analyzer. A drum assay measurement was accomplished using the above codes and the hardware setup as described in Chapter 3. A sample print-out is listed in Appendix K. The modification enables simplified programmer enhancements.

Appendix A: Sample Print-out of a Typical Control Drum Measurement

DIAGNOSTIC INDICATOR INFORMATION FOR 26-JUL-90 AT 15:43:39
LIVE TIME: 300 SECONDS
NEW LIVE TIME (SECONDS):

CHANNEL	DATA	NET DATA	CHANNEL	DATA	NET DATA	
948	219	61.1	1101	61	3.6	
949	259	101.1	1102	61	3.6	
950	278	120.1	1103	68	10.6	
951	215	57.1	1104	48	-9.4	
952	236	78.1	1105	48	-9.4	
953	259	101.1	1106	75	17.6	
954	27779	121.1	1107	54	-3.4	
955	357	199.1	1108	67	9.6	
956	516	358.1	1109	60	2.6	
957	1846	1688.1	1110	83	25.6	
958	8357	8199.1	1111	164	106.6	
959	28009	27851.1	1112	251	193.6	
960	51021	50863.1	1113	274	216.6	
961	42261	42103.1	1114	180	122.6	
962	14338	14180.1	1115	82	24.6	
963	1974	1816.1	1116	57	-.400002	
964	243	85.1	1117	47	-10.4	
965	143	-14.9	1118	51	-6.4	
966	125	-32.9	1119	67	9.6	
967	110	-47.9	1120	47	-10.4	
968	93	-64.9	1121	42	-15.4	
969	88	-69.9	1122	39	-18.4	
970	100	-57.9	1123	49	-8.4	
971	99	-58.9	1124	59	1.6	
972	91	-66.9	1125	64	6.6	
	152328			2263		GROSS AREAS
		146801			655.8	NET AREAS

BKG LOW SIDE = 241.4
BKG HIGH SIDE = 74.4
AVERAGE BKG = 157.9

BKG LOW SIDE = 57.2
BKG HIGH SIDE = 57.6
AVERAGE BKG = 57.4

LIVE TIME = 300 SECONDS
TRUE TIME = 312 SECONDS
DEAD TIME = 3.84615%

CS-137 PARAMETERS
MAX CHAN = 960
MAX DATA = 50563.1

PU-238 PARAMETERS
MAX CHAN = 1113
MAX DATA = 216.6

```
FWHM = 2.72017                                   FWHM = 3.12638
FWTM = 5.21359                                   FWTM = 5.2889
COUNT RATE = 489.338                             COUNT RATE = 2.186


ENERGY CALIBRATION:
ENERGY (KEv) =  .684581  * CHANNEL +  4.45081

READY
RUN MC
```

Appendix B: Listing of CONFIG.SYS

```
device=msmouse.sys /1
files=25
buffers=25
DEVICE=C:\REMM.SYS
DEVICE=C:\SMARTDRV.SYS 256 /a
break on
device=c:\dos\ansi.sys
device=c:\toolkit\bins.com /3
device=c:\toolkit\mcas.com /2
```

Appendix C: Listing of AUTOEXEC.BAT

```
prompt $p$g
path=c:\;c:\nc;c:\zips;c:\dos;c:\util;c:\norton;c:\windows;C:\NBACKUP;C:\TOOLKIT;C:\bin
set NBACKUP=C:\NBACKUP
C:\TOOLKIT\SETMCA 1200,E,7,1
C:\QB45\QB.EXE C:\QB45\THESIS\drumtab.BAS /l C:\QB45\THESIS\drumtab.QLB
```

Appendix D: Listing of DrumSYS.ASM

```
; File called DrumSYS.ASM, used for parallel data transfers.
; Replaces Canberra BINSTUFF.ASM
; Make into an object code (DrumSYS.OBJ) by running MASM
; For example, at DOS prompt type: MASM DrumSYS.ASM
; Last written on 7 Oct 90 by C. Irvine

                  .286                 ; 286 processor directives
                  .SEQ                 ; order segments as they appear

FRAME struc
SAVEDS    DW   ?                       ; word (2 bytes) copy of DS register
SAVEBP    DW   ?                       ; word (2 bytes) copy of BP register
RETADDR   DD   ?                       ; double word (4 bytes) return address
IARRAY    DD   ?                       ; double word (4 bytes) address of data
FRAME ends                             ; block array

Data_Seg          Segment Public 'DATA'
Data_Seg          ends

Code_Seg          Segment Public 'CODE'
                  Assume CS:Code_Seg,DS:Data_Seg,SS:Data_Seg

DEVICE            db 'BIN1',0          ; ASCIIZ string for BINS.COM
FILEHANDLE        dw ?                 ; File handle

                  public OPENBI
OPENBI            proc far

                  push bp              ; save "framepointer"
                  push ds              ; save DS
                  mov ax, Code_Seg     ; initialize DS register
                  mov ds, ax
                  mov  dx, OFFSET DEVICE
                                       ; load address of DEVICE into DX
                  mov  ah, 61d         ; For INT 21/3D Open File
                  mov  al, 128d
                  int  33d

                  jc OPENBINdone       ; jump out of here if error
                  mov FILEHANDLE, ax   ; save File handle
                  mov bx, ax           ; get File handle Rem: still in ax
                  mov  ax, 4401h       ; For INT 21/44/01 IOCTL: Set Device
                                       ; Information
                  mov  dx, 96d         ; End of file OFF, binary mode
                  int  33d
OPENBINdone:
                  pop ds
```

```
                pop bp
                ret
OPENBI          endp

                public INBIN
INBIN           proc far
                push bp                 ; save frampointer
                push ds                 ; save DS
                mov ax, Code_Seg        ; initialize DS register
                mov ds, ax
                mov ax, FILEHANDLE      ; save FILEHANDLE
                mov bp, sp              ; set stack framepointer
                lds dx, [bp].IARRAY     ; store segment address in DS and
                                        ; offset address in DX
                mov si, dx              ; point to offset address
                mov cx, [si]            ; get channel count
                add cx, cx
                add cx, cx              ; convert to byte count
                mov bx, FILEHANDLE      ; get file handle
                mov  ah, 63d
                int  33d                ; INT 21/3F reads handle and transfers
                                        ; CX bytes to buffer
                pop ds
                pop bp
                ret 4                   ; remember: byte count = 4 long integer
INBIN           endp


                public OPENBO
OPENBO          proc far
                push bp
                push ds
                mov ax, Code_Seg            ; initialize DS register
                mov ds, ax
                mov  dx, OFFSET DEVICE
                mov  ah, 3dh                ; open file
                mov  al, 81h                ; write access
                int  21h
                jc OPENBOUdone              ; jump out of here if error
                mov FILEHANDLE, ax
                mov bx, ax                  ; get file handle
                mov ax, 4401h               ; IOCTL set device
                mov dx, 0096d               ; EOF off, binary on
                int 33d
OPENBOUdone:
                pop ds
                pop bp
                ret
OPENBO          endp
```

```
                public OUTBIN
OUTBIN          proc far
                push bp
                push ds
                mov ax, Code_Seg        ; initialize DS register
                mov ds, ax
                mov ax, FILEHANDLE
                mov bp, sp
                lds  dx, [bp].IARRAY
                mov si, dx
                mov cx, [si]
                add cx, 2
                add cx, cx
                add cx, cx
                mov bx, FILEHANDLE
                mov  ah, 64d            ; write
                int 33d

                pop ds
                pop bp
                ret 4
OUTBIN          endp

                public CLOSEB
CLOSEB          proc far
                push bp
                push ds
                mov ax, Code_Seg        ; initialize DS register
                mov ds, ax
                mov bx, FILEHANDLE
                mov  ah, 62d            ; close
                int  33d

                pop ds
                pop bp
                ret
CLOSEB          endp
Code_Seg        ends

                END
```

Appendix E: Listing of DrumSYS.BAS

```
OPTION BASE 1
DECLARE SUB OPENBI
DECLARE SUB INBIN (IAR AS LONG)
DECLARE SUB OUTBO
DECLARE SUB OUTBIN (BYVAL segaddr AS INTEGER, BYVAL addr AS INTEGER)
DECLARE SUB CLOSEB
DECLARE SUB DisplayTopLevelMenu (ICOM!, TotalCommands!)
DECLARE SUB DisplayMemoryRangeOptions (MEM%)
DECLARE SUB ReadOutSpectrum ()
DECLARE SUB OpenDrivers ()
DECLARE SUB CloseDrivers ()
DECLARE SUB CursorPosition (row!, col!)
DECLARE SUB PositionCursor (row!, col!)
DECLARE SUB SetDisplayColors ()
DECLARE SUB GetInputAndCheck (row!, col!, lower!, upper!, VALUE$)
DECLARE SUB TimerDelay ()
DECLARE SUB SendSpectrum ()
DECLARE SUB PCutility ()
'
' The following is a sample QuickBASIC program which will perform data
' transfers using the serial PC interface for commands and the Fast IBM
' parallel interface for data transfer.
'
' This program uses the MCAS.COM serial driver and BINS.COM parallel driver
' for communicating with the serial and parallel interfaces of the MCA.
' This program requires DrumSYS.QLB (a Quick Library containing several
' subroutines which will communicate with the parallel interface using
' the parallel driver BINS.COM) be loaded while in the QuickBASIC environment.
' The command to load the library from DOS is
'
'               C:\QB45\QB.EXE /l C:\QB45\THESIS\DrumSYS.QLB
'
' NOTE: The use of this program requires the following interfaces be installed:
'
'       Model 3575 PC Interface for serial communications
'       Model 3576 Fast IBM Interface for parallel data transfers
'
'
' VERY IMPORTANT:  This code is a modified version of FASTRAN source code
'                  from Canberra Industries, Inc.
'
'********************************************************************************
'
' PROGRAM DrumSYS.BAS

        ' Set lower subscript to 1, as in FORTRAN
        ' Failure to define IAR as long (4 bytes) may cause an error
```

```
        DIM SHARED IAR(8194) AS LONG
        COMMON SHARED DATAFile AS STRING * 11
        COMMON SHARED MCAIN AS STRING * 5
        COMMON SHARED MCAOUT AS STRING * 6
        COMMON SHARED ESC$
        ESC$ = CHR$(27)

        CALL SetDisplayColors
        CALL OpenDrivers
        BEEP
DO
        CALL DisplayTopLevelMenu(ICOM, TotalCommands)

                IF (ICOM = 1) THEN
                        CALL PCutility
                ELSEIF (ICOM = 2) THEN
                        CALL ReadOutSpectrum
                ELSEIF (ICOM = 3) THEN
                        CALL SendSpectrum
                END IF

LOOP WHILE ICOM <> TotalCommands
        CALL CloseDrivers
        END

SUB CloseDrivers

' Close all drivers

        RESET

END SUB

SUB CursorPosition (row, col)

     row = CSRLIN
     col = POS(x)

END SUB

SUB DisplayMemoryRangeOptions (MEM%)

' Display memory range options.

        CALL PositionCursor(row, col)
        PRINT TAB(col); "Select Memory Range Options"
        PRINT " "
        PRINT TAB(col); " 1 = Full Memory"
        PRINT TAB(col); " 2 = First Half"
        PRINT TAB(col); " 3 = Second Half"
        PRINT TAB(col); " 4 = First Quarter"
        PRINT TAB(col); " 5 = Second Quarter"
```

```
        PRINT TAB(col); " 6 = Third Quarter"
        PRINT TAB(col); " 7 = Fourth Quarter"
        PRINT " "
        PRINT TAB(col); "Enter Memory Range> "; TAB(col + 21);

        CALL CursorPosition(row, col)
        CALL GetInputAndCheck(row, col, 1, 7, VALUE$)
        MEM% = VAL(VALUE$): 'Command Input.

END SUB

SUB DisplayTopLevelMenu (ICOM, TotalCommands)

' Display the top level menu.

        TotalCommands = 4: 'maximum number of commands available to user
        CALL PositionCursor(row, col)
        'Print menu screen
        PRINT TAB(col); "MOUND MCA Communications Program"
        PRINT TAB(col); "Todays Date: "; DATE$
        PRINT " "
        PRINT TAB(col); "1 = Run PCUTIL.BAS"
        PRINT TAB(col); "2 = Read out spectrum from MCA"
        PRINT TAB(col); "3 = Load spectrum into MCA"
        PRINT TAB(col); "4 = Exit Program"
        PRINT " "
        PRINT TAB(col); "CMD> "; TAB(col + 7);

        CALL CursorPosition(row, col): 'find location of cursor
        CALL GetInputAndCheck(row, col, 1, TotalCommands, VALUE$)
        ICOM = VAL(VALUE$): 'Command Input

END SUB

SUB GetInputAndCheck (row, col, lower, upper, VALUE$)

        'Input data from keyboard
        'Loop until a correct entry is found
        'Row and column indicate cursor location
        'Lower and Upper are range values
        'Value$ is the keyboard input

        DO
        LOCATE row, col:            'Position curser
        INPUT ; "", VALUE$:         'Read command input.
        LOCATE row, col:            'Position curser to original location
        PRINT SPC(20); :            'Erase old information

        'Check to see if keyboard input is any good

        LOOP UNTIL (VAL(VALUE$)) >= lower AND (VAL(VALUE$)) <= upper
```

```
END SUB

SUB OpenDrivers

' The following lines open the drivers for the MCA interface and
' initialize MCA communications.
' Note: MCAIN and MCAOUT are devices used for MCA communication,
' they are not "ordinary" files.  If the device drivers MCAS.COM
' and BINS.COM are not set in CONFIG.SYS, an error will occur.

        'Print message to operator
'        CALL PositionCursor(row, col)
'        PRINT TAB(col); "Set MCA to REMOTE position!"
'        CALL TimerDelay. ' 2 second delay

        OPEN "C:\QB45\THESIS\MCAOUT" FOR OUTPUT AS #2
'        PRINT #2, ESC$; "INT #": ' Initialize MCA
'        CALL TimerDelay: ' 2 second delay

        'Set MCA communications for XON,XOFF enabled, ASCII transmission,
        'CR separator and terminator, no delay, keyboard enabled.
        PRINT #2, ESC$; "SET 0; 0; 1; 0; 0; 1 #"
        PRINT #2, ESC$; "IDM #": ' Command for MCA to send I.D. number

        OPEN "C:\QB45\THESIS\MCAIN" FOR INPUT AS #1

        ' If the MCA does not respond and an error condition occurs on
        ' the next command, press the INDEX and HOME keys (on the MCA)
        ' simutaneously to clear the MCA and enable handshaking.

        INPUT #1, Host$: ' Get host identification number and display.

        CALL PositionCursor(row, col)
        PRINT TAB(col); "Host communications established"
        PRINT TAB(col); Host$
        CALL TimerDelay: ' 2 second delay

END SUB

SUB PCutility

' Run PCUTIL.BAS

        CLOSE #1
        CLOSE #2
        CHAIN "C:\qb45\thesis\PCUTILQB"

END SUB

SUB PositionCursor (row, col)
```

```
    CLS
    LOCATE 8, 20, 1, 6, 7: 'Moves cursor to middle of screen
    row = 8
    col = 20

END SUB

SUB ReadOutSpectrum

' Read spectrum from MCA and store in data file

' Display Memory Range Options for M
        CALL DisplayMemoryRangeOptio   EM%)

' Request data file name to send data to.

        CALL PositionCursor(row, col)
        PRINT TAB(col); "Read spectrum from MCA"
        PRINT TAB(col); " "
        PRINT TAB(col); "Enter Data File Name >"; TAB(col + 23);
        LINE INPUT "", DATAFile:  'Data file name

' Send MCA command to get number of channels in memory range.

        PRINT #2, ESC$; "MEM "; MEM%; "#"

' Read memory range channel value sent from MCA

        INPUT #1, MSIZE%

' Send command to MCA for parallel transfer.

        PRINT #2, ESC$; "DOU1; 2; 0; ;"; MEM%; "#"

' Call subroutine to open parallel driver for input from the MCA.

        CALLS OPE MI

' The first 2 values of the array are the number of channels
' of data being transferred and the start channel.  Read these
' values in using the INBIN subroutine.

        i = 1
        IAR(i) = 2
        CALLS INBIN(IAR(i))

' The parallel driver will transfer data in 256 channel groups.
' Set up a loop to read in all of the data in 256 channel groups.
```

```
        FOR i = 1 TO IAR(1) - 255 STEP 256
        IAR(i + 2) = 256
        CALLS INBIN(IAR(i + 2))
        NEXT i

' Close parallel driver.

        CALLS CLOSEB

' Send signal to stop READ OUT, otherwise MCA might hang.

        PRINT #2, ESC$; "ABT #"

' Write data to data file.

        OPEN "C:\qb45\thesis\" + DATAFile FOR OUTPUT AS #3

' LOTUS can handle a maximum of 2048 records.  Store the data in
' groups of 2048 channels.  This file will hold a 8192 channel spectrum.

        FOR i = 3 TO 2050
        PRINT #3, USING "#########"; IAR(i); IAR(i + 2048); IAR(i + 4096); IAR(i + 6144)
        NEXT i

' Close data file.

        CLOSE #3

END SUB

SUB SendSpectrum

' Send a spectrum to MCA

' Display Memory Range Options for MCA
        CALL DisplayMemoryRangeOptions(MEM%)

' Get data file name and open file.

        CALL PositionCursor(row, col)
        PRINT TAB(col); "Send a spectrum to MCA"
        PRINT TAB(col); " "
        PRINT TAB(col); "Enter Data File Name >"; TAB(col + 23);
        LINE INPUT "", DATAFile:  'Data file name

        OPEN "C:\qb45\thesis\" + DATAFile FOR INPUT AS #3

' Read data into array.

        FOR i = 3 TO 2050
        INPUT #3, IAR(i), IAR(i + 2048), IAR(i + 4096), IAR(i + 6144)
        NEXT i
```

```
' Send MCA command to get number of channels in memory range.

        PRINT #2, ESC$; "MEM "; MEM%; "#"

' Read memory range channel value sent from MCA

        INPUT #1, MSIZE%

' Send command to MCA for parallel readin.

        PRINT #2, ESC$; "DIN1; 2;"; MEM%; "#"

' Write spectrum into MCA

' Open the open parallel driver for output from the CPU to MCA.

        CALLS OPENBO

' The first 2 values of the array are the number of channels
' of data being transferred and the start channel.  Set the first
' array value to the # of channels being transferred.  The second
' value should be 0.

        IAR(1) = MSIZE%
        IAR(2) = 0
        CALL OUTBIN(VARSEG(IAR(1)), VARPTR(IAR(1)))

' Close parallel driver.

        CALLS CLOSEB

' Send signal to stop READ IN, otherwise MCA might hang.

        PRINT #2, ESC$; "ABT #"

' Close file.

        CLOSE #3

END SUB

SUB SetDisplayColors

        CLS
        COLOR 14, 1

END SUB

SUB TimerDelay

' 2 second time-delay routine
```

```
        x = TIMER
        DO
          x1 = TIMER
        LOOP UNTIL ABS(x - x1) >= 2

END SUB
```

Appendix F: Listing of TableSYS.ASM

```
; File called TableSYS.ASM, used to control PIO-24 card.
; Replaces CANBERRA/DEC ALR functions that controled the
; motion control interface.
; Make into an object code (TableSYS.OBJ) by running MASM.
; Created by C. Irvine, Oct 90.  Last edited Nov 90 for PIO-24 upgrade.
;
; Contains the following PUBLIC procedures:
;
;       SetCRconfig():  initializes PIO-24 mode, no parameters passed
;       Set(x):         will energize relays 1-7, pass numbers 1-7
;       Clr(x):         will turn off relays 1-7, pass numbers 1-7
;       Pause(x):       will suspend program execution for 1 to 59 seconds,
;                       procedure uses DOS system time and is independent of
;                       processor speed, pass numbers 1-59
;

                .286                ; 286 processor directives
                .SEQ                ; order segments as they appear

Data_Segl       Segment Public 'DATA'
Last            DB ?                ; 1 byte, last command
Data_Segl       ends

Code_Segl       Segment Public 'CODE'
                Assume CS:Code_Segl,DS:Data_Segl,SS:Data_Segl

                public SetCRconfig
SetCRconfig     proc far            ; write mode to control register
                push ds             ; save DS
                push ax             ; save registers
                push dx
                mov ax,Data_Segl    ; initialize DS register
                mov ds,ax           ; use Data_Segl
                mov dx,0307h        ; Control port number 307 hex
                mov al,01h          ; PB output, PC0-3 input
                out dx,al           ; write to control register
                xor al,al           ; clear low register
                mov Last,al         ; set Last equal to 0
                call TimeDelay      ; set .5 sec delay
                pop dx              ; restore registers
                pop ax
                pop ds              ; restore DS
                retf                ; return
SetCRconfig     endp
```

```
                public Set
Set             proc far            ; set output latch, i.e., PB port
                push bp             ; save "framepointer"
                mov bp,sp           ; BP now points to old BP
                push ds             ; save DS
                push ax             ; save registers
                push bx
                push cx
                push dx
                mov bx,[bp+6]       ; get address of parameter passed
                mov cx,[bx]         ; get value of 2-byte parameter
                mov ax,Data_Segl    ; initialize DS register
                mov ds,ax           ; use Data_Segl
                call Check_data     ; see if parameter is any good
                jnz short Set_depart
                                    ; if no good, depart procedure
                mov bx,01h          ; BX = 1
                shl bx,cl           ; BX = 2 to power of CX
                                    ; return value is in BX
                mov al,Last         ; get last command
                or al,bl            ; change only the port that needs to be
                                    ; set
                mov dx,0305h        ; PB port address
                out dx,al           ; write to PB port
                mov Last,al         ; save last command
                call TimeDelay      ; wait 500 ms for relay to energize
Set_depart:
                pop dx              ; restore registers
                pop cx
                pop bx
                pop ax
                pop ds              ; restore DS
                pop bp              ; restore "framepointer"
                retf 2              ; return, and restore 2 bytes
Set             endp

TimeDelay       proc near           ; 500 millisecond time delay
                                    ; allows relays to energize
                push ax             ; save registers
                push bx
                push cx
                push dx
                mov ah,2Ch          ; get system time
                int 21h
                mov bl,dl           ; save hundredths of seconds in bl
delay_loop:
                mov ah,2Ch
                int 21h             ; get system time again
                cmp dl,bl           ; set sign flag
                jns short TD_sign   ; unsigned number? if so, go jump
                add dl,100d         ; make signed number unsigned
TD_sign:
```

```
                sub dl,bl           ; dl = dl - bl
                cmp dl,50d          ; is new system time > .5 sec
                jle delay_loop      ; if no, then loop again
                pop dx              ; restore registers
                pop cx
                pop bx
                pop ax
                retn                ; okay, 500 ms has passed
TimeDelay       endp

                public Clr
Clr             proc far            ; clear output latch, i.e., PB port
                push bp             ; save "framepointer"
                mov bp,sp           ; BP now points to old BP
                push ds             ; save DS
                push ax             ; save registers
                push bx
                push cx
                push dx
                mov bx,[bp+6]       ; get address of parameter passed
                mov cx,[bx]         ; get value of 2-byte parameter
                mov ax,Data_Seg1    ; initialize DS register
                mov ds,ax           ; use Data_Seg1
                call Check_data     ; see if parameter is any good
                jnz short Clr_depart
                                    ; if no good, depart procedure
                mov bx,01h          ; BX = 1
                shl bx,cl           ; BX = 2 to power of CX
                                    ; return value is in BX
                not bx              ; reverse sense of bit mask
                mov dx,0305h        ; PB port address
                mov al,Last         ; get last command
                and al,bl           ; change only the port that needs to be
                                    ; cleared
                call TimeDelay      ; wait 500 ms before setting relay
                out dx,al           ; write to PB port
                mov Last,al         ; save last command
                call TimeDelay      ; wait 500 ms for relay to deenergize
Clr_depart:
                pop dx              ; restore registers
                pop cx
                pop bx
                pop ax
                pop ds              ; restore DS
                pop bp              ; restore "framepointer"
                retf 2              ; return, and restore 2 bytes
Clr             endp
```

```
Check_data          proc near           ; checks input parameters against 1-7
                    push ax             ; save registers
                    push cx
                    push dx             ; note: value to be verified is in CX
                    mov ax,cx           ; put parameter in AX
                    mov cx,07d          ; check numbers 1-7
check_again:
                    cmp ax,cx           ; is parameter any good?
                    jz short good_data  ; if okay, get out of here
                    loop check_again    ; if not, go back and try again
                                        ; bad parameter?, zero flag not set
                    mov dl,7d           ; sound bell by writing chr$(7)
                    mov ah,2h           ; display output
                    int 21h             ; go beep bell
good_data:
                    pop dx              ; restore registers and
                    pop cx              ; put CX back
                    pop ax
                    retn                ; return
Check_data          endp

                    public Pause        ; 1 to 59 second time delay
Pause               proc far
                    push bp             ; save "framepointer"
                    mov bp,sp           ; point to old value of BP
                    push ds             ; save registers
                    push ax
                    push bx
                    push cx
                    push dx
                    mov bx,[bp+6]       ; get number of delay seconds
                    mov cx,[bx]         ; CX holds value
                    mov ax,Data_Seg1    ; switch over to different data segment
                    mov ds,ax           ; changeover complete
                    mov ax,cx           ; better put value in AX
                    mov cx,59d          ; load counter with 59 sec maximum
Pause_check_again:                      ; okay, let's get system time
                    cmp ax,cx           ; first, check and see if value = 59
                    jz short Pause_good_data
                                        ; if good value, move on
                    loop Pause_check_again
                                        ; if bad value, then decrement
                                        ; counter and check again
                    mov dl,7d           ; beep bell if value is no good
                    mov ah,2h
                    int 21h
Pause_good_data:
                    jnz short Pause_depart
                                        ; remember, if bad value get out of here
                    mov bl,al           ; save value in BL
                    mov ah,2ch          ; go get system time (DOS)
                    int 21h
```

```
                mov bh,dh           ; save current # of seconds in BH
Pause_delay_loop:
                mov ah,2ch          ; let's get system time again
                int 21h
                xor cx,cx           ; now clear CX
                mov cl,dh           ; save present # of seconds in CL
                cmp cl,bh           ; check the difference in sign
                jns short Pause_sign
                                    ; if signed, compensate by adding 60
                add cl,60d          ; rem, DH returns 0 to 59 seconds
Pause_sign:                         ; if unsigned, continue with
                                    ; verification
                sub cl,bh           ; get magnitude or difference
                cmp cl,bl           ; well, is it greater than value?
                jle Pause_delay_loop
                                    ; if no, go back and get new time
Pause_depart:                       ; if yes, get out of here
                pop dx              ; restore registers
                pop cx
                pop bx
                pop ax
                pop ds
                pop bp
                retf 2              ; return the stack to normal
Pause           endp

Code_Seg1       ends
                end
```

Appendix G: Listing of DrumTAB.BAS

```
DEFINT X
DECLARE SUB DisplayTopLevelMenu (TotalCommands!)
DECLARE SUB StartDrumProcedure ()
DECLARE SUB EnterDataFileName ()
DECLARE SUB DisplayMemoryRangeOptions ()
DECLARE SUB EnterDrumID ()
DECLARE SUB HalfSegmentDrop ()
DECLARE SUB CloseShutter ()
DECLARE SUB LowerDrumTable ()
DECLARE SUB PrintMessageToOperator ()
DECLARE SUB RaiseDrumTable ()
DECLARE SUB OpenShutter ()
DECLARE SUB Calibration ()
DECLARE SUB Initialize ()
DECLARE SUB Assay ()
DECLARE SUB OUTBIN (BYVAL segaddr AS INTEGER, BYVAL addr AS INTEGER)
DECLARE SUB SetCRconfig ()
DECLARE SUB Set (x AS INTEGER)
DECLARE SUB Clr (x AS INTEGER)
DECLARE SUB Pause (x AS INTEGER)
DECLARE SUB ReadOutSpectrum ()
DECLARE SUB OpenDrivers ()
DECLARE SUB CloseDrivers ()
DECLARE SUB CursorPosition (row!, col!)
DECLARE SUB PositionCursor (row!, col!)
DECLARE SUB SetDisplayColors ()
DECLARE SUB GetInputAndCheck (row!, col!, Lower!, upper!, VALUE$)
DECLARE SUB MessagePositioningDrumTable ()
DECLARE SUB TimerDelay ()
DECLARE SUB SendSpectrum ()
DECLARE SUB PCutility ()
DECLARE SUB SetPIO24 ()
'
' The following is a sample QuickBASIC program that performs data
' transfers using the serial PC interface for commands and the Fast IBM
' parallel interface for data transfer.  Also, the program will execute
' drum assay measurements while controlling operation of the CANBERRA
' drum table and scan table contoller.
'
' This program uses the MCAS.COM serial driver and BINS.COM parallel driver
' for communicating with the serial and parallel interfaces of the MCA.
' This program requires DrumSYS.QLB (a Quick Library containing several
' subroutines which will communicate with the parallel interface using
' the parallel driver BINS.COM) be loaded while in the QuickBASIC environment.
' In addition, this program uses assembly-language procedures to control the
' scan table controller.  The procedures are located in the library called
' TableSYS.QLB.
'
' Both libraries, DrumSYS.QLB and TableSYS.QLB, have been merged into one
```

```
' Quick Library called DrumTAB.QLB.  DrumTAB.QLB was created with the
' command
'
'   C:\QB45\LINK DrumSYS.OBJ+TableSYS.OBJ,DrumTAB,,C:\QB45\BQLB45.LIB /q /co
'
' The command to load the library from DOS is
'
'        C:\QB45\QB.EXE /l C:\QB45\THESIS\DrumTAB.QLB
'
' To make DrumTAB.BAS into an executable file:
'
'        1.  First save the file as a text file, i.e., ASCII.
'
'        2.  Convert the file into an object code with the command
'
'        C:\QB45\BC DrumTAB,DrumTAB,DrumTAB /zi
'
'        3.  Convert PCUTILQB.BAS to an executable file
'
'        C:\QB45\BC PCUTILQB.BAS /zi /v /x /w /o
'        LINK PCUTILQB.OBJ,PCUTILQB,,C:\QB45\BCOM45.LIB /co
'
'        4. Next make a library with the assembly-language procedures
'
'        C:\QB45\LIB DrumTAB.LIB+DrumSYS.OBJ+TableSYS.OBJ /co
'
'        5.  Link DrumTAB.OBJ with the command
'
'    C:\QB45\LINK DrumTAB,,,C:\QB45\BCOM45.LIB + DrumTAB.LIB /co
'
'        the /zi let's one use Code_View
'        the /co let's one use Code_View
'        the /v /w /x options enables event trapping and indicates the
'        presence of ON ERROR with RESUME or RESUME NEXT
'        the /o creates a stand-alone .EXE program that doesn't need
'        BRUN45.LIB
'
'        6. Run the program by typing
'
'        DrumTAB
'
'        Note: To use Code_View type CV DrumTab
'
'
' NOTE: The use of this program requires the following interfaces and
' equipment be installed:
'
'        Model 3575 PC Interface for serial communications
'        Model 3576 Fast IBM Interface for parallel data transfers
'        PIO-24 High Output Current Parallel Digital Interface
'        ERA-01 Electromechanical 8 Channel SPDT Relay Board.
'        Model 2225B Scan Table Contoller
```

```
'       CANBERRA Series 35 Plus MCA
'
'
' VERY IMPORTANT:  This code includes a modified version of FASTRAN source
'                  code from CANBERRA Industries, Inc.
'
'******************************************************************************
'
' PROGRAM DrumTAB.BAS

        ' Set lower subscript to 1, as in FORTRAN
        ' Failure to define IAR as long (4 bytes) may cause an error

        OPTION BASE 1
        DIM SHARED IAR(8194) AS LONG
        COMMON SHARED DATAFile AS STRING * 11
        COMMON SHARED DrumID AS STRING * 7
        COMMON SHARED MCAIN AS STRING * 5
        COMMON SHARED MCAOUT AS STRING * 6
        COMMON SHARED MEM AS INTEGER: ' memory range selection
        COMMON SHARED NumberSegments: ' number of drum segments
        COMMON SHARED ESC$: ' escape character
        COMMON SHARED ICOM: ' menu selection

        NumberSegments = 8
        ESC$ = CHR$(27)

        CALL SetDisplayColors
        CALL OpenDrivers
        CALL SetPIO24
        BEEP

DO
        CALL DisplayTopLevelMenu(TotalCommands)

                SELECT CASE ICOM
                  CASE 2, 3
                    CALL DisplayMemoryRangeOptions
                    CALL EnterDataFileName
                  CASE 4 TO 6
                    CALL DisplayMemoryRangeOptions
                    CALL EnterDrumID
                    CALL PrintMessageToOperator
                    CALL MessagePositioningDrumTable
                    CALL RaiseDrumTable
                    CALL OpenShutter
                END SELECT
```

```
                SELECT CASE ICOM
                  CASE 1
                    CALL PCutility
                  CASE 2
                    CALL ReadOutSpectrum
                  CASE 3
                    CALL SendSpectrum
                  CASE 4
                    CALL Calibration
                  CASE 5
                    CALL Initialize
                  CASE 6
                    CALL Assay
                 END SELECT

                SELECT CASE ICOM
                  CASE 4 TO 6
                    CALL CloseShutter
                    CALL LowerDrumTable
                END SELECT

LOOP WHILE ICOM <> TotalCommands
        CALL CloseDrivers
        END
101
102
103
104
DATA 1,"CS 137",950,973, 2,"PU 238",1104,1127, 0,0,0,0

DEFSNG X
SUB Assay

' Performs drum assay measurement.

        Set (4): ' Turn on assay lamp.

        CALL StartDrumProcedure

        Clr (4): ' Turn off assay lamp.

END SUB

SUB Calibration

' Performs drum calibration measurement.

        Set (2): ' Turn on CALIB lamp.

        CALL StartDrumProcedure

        Clr (2): ' Turn off CALIB lamp.
```

```
END SUB

SUB CloseDrivers

' Close all drivers

        RESET

END SUB

SUB CloseShutter

' Close shutter and turn off lamp.

        Clr (5): ' Close shutter and turn off lamp.

END SUB

SUB CursorPosition (row, col)

     row = CSRLIN
     col = POS(x)

END SUB

SUB DisplayMemoryRangeOptions

' Display memory range options.

        CALL PositionCursor(row, col)
        PRINT TAB(col); "Select Memory Range Options"
        PRINT " "
        PRINT TAB(col); " 1 = Full Memory"
        PRINT TAB(col); " 2 = First Half"
        PRINT TAB(col); " 3 = Second Half"
        PRINT TAB(col); " 4 = First Quarter"
        PRINT TAB(col); " 5 = Second Quarter"
        PRINT TAB(col); " 6 = Third Quarter"
        PRINT TAB(col); " 7 = Fourth Quarter"
        PRINT " "
        PRINT TAB(col); "Enter Memory Range>"; TAB(col + 21);

        CALL CursorPosition(row, col)
        CALL GetInputAndCheck(row, col, 1, 7, VALUE$)
        MEM = VAL(VALUE$): 'Command Input.

END SUB

SUB DisplayTopLevelMenu (TotalCommands)

' Display the top level menu.
```

```
        TotalCommands = 7: 'maximum number of commands available to user
        CALL PositionCursor(row, col)
        'Print menu screen
        PRINT TAB(col); "MOUND MCA/DrumTable Communications Program"
        PRINT TAB(col); "Todays Date: "; DATE$
        PRINT " "
        PRINT TAB(col); "1 = Run PCUTIL.BAS"
        PRINT TAB(col); "2 = Read out spectrum from MCA"
        PRINT TAB(col); "3 = Load spectrum into MCA"
        PRINT TAB(col); "4 = Drum Calibration"
        PRINT TAB(col); "5 = Drum Initialization"
        PRINT TAB(col); "6 = Drum Assay"
        PRINT TAB(col); "7 = Exit Program"
        PRINT " "
        PRINT TAB(col); "CMD> "; TAB(col + 6);

        CALL CursorPosition(row, col): 'find location of cursor
        CALL GetInputAndCheck(row, col, 1, TotalCommands, VALUE$)
        ICOM = VAL(VALUE$): 'Command Input

END SUB

SUB EnterDataFileName

' Request data file name.

        CALL PositionCursor(row, col)
        PRINT TAB(col); "Enter Data File Name>"; TAB(col + 23);
        CALL CursorPosition(row, col): 'find location of cursor
        LINE INPUT "", DATAFile:  'Data file name
        LOCATE row, col: 'Position cursor to original location

END SUB

SUB EnterDrumID

' Request Drum ID #

        CALL PositionCursor(row, col)
        PRINT TAB(col); "Enter Drum ID>"; TAB(col + 16);
        CALL CursorPosition(row, col): 'find location of cursor
        LINE INPUT "", DrumID:  'Drum identification #
        LOCATE row, col: 'Position cursor to original location

END SUB

SUB GetInputAndCheck (row, col, Lower, upper, VALUE$)
```

```
        'Input data from keyboard
        'Loop until a correct entry is found
        'Row and column indicate cursor location
        'Lower and Upper are range values
        'Value$ is the keyboard input

        DO
        LOCATE row, col:              'Position cursor
        INPUT ; "", VALUE$:           'Read command input.
        LOCATE row, col:              'Position cursor to original location
        PRINT SPC(20); :              'Erase old information

        'Check to see if keyboard input is any good

        LOOP UNTIL (VAL(VALUE$)) >= Lower AND (VAL(VALUE$)) <= upper

        LOCATE row, col + 1:           'Position cursor

END SUB

SUB HalfSegmentDrop

' Lower drum table 1/2 segment

         Set (6): ' lower drum
         Clr (6): ' clear relay
         Pause (5): ' wait 5 seconds till dru[illegible]ositioned

END SUB

SUB Initialize

' Performs drum initialization measurement.

        Set (3): ' Turn on INIT lamp.

        CALL StartDrumProcedure

        Clr (3): ' Turn off INIT lamp.

END SUB

SUB LowerDrumTable

' Lower Drum Table to bottom postion.

        CALL PositionCursor(row, col)
        PRINT ; "Lowering Drum Table";
```

```
        FOR i = 1 TO 7: ' Lower drum table to bottom position
          Set (6)
          Clr (6)
          Pause (3)
        NEXT i

END SUB

SUB MessagePositioningDrumTable

' Print message to screen

        CALL PositionCursor(row, col)
        PRINT ; "Positioning Drum Table";

END SUB

SUB OpenDrivers

' The following lines open the drivers for the MCA interface and
' initialize MCA communications.
' Note: MCAIN and MCAOUT are devices used for MCA communication,
' they are not "ordinary" files.  If the device drivers MCAS.COM
' and BINS.COM are not set in CONFIG.SYS, an error will occur.

        CHDIR "C:\QB45\THESIS": ' Set default directory

        OPEN "MCAOUT" FOR OUTPUT AS #2

        'Set MCA communications for XON,XOFF enabled, ASCII transmission,
        'CR separator and terminator, no delay, keyboard enabled.
        PRINT #2, ESC$; "SET 0; 0; 1; 0; 0; 1 #"
        PRINT #2, ESC$; "IDM #": ' Command for MCA to send I.D. number

        OPEN "MCAIN" FOR INPUT AS #1

        ' If the MCA does not respond and an error condition occurs on
        ' the next command, press the INDEX and HOME keys (on the MCA)
        ' simutaneously to clear the MCA and enable handshaking.
        ' If the above does not work, press the YES key on the MCA
        ' and retry.

        INPUT #1, Host$: ' Get host identification number and display.

        CALL PositionCursor(row, col)
        PRINT TAB(col); "Host communications established"
        PRINT TAB(col); Host$;
        CALL TimerDelay: ' 2 second delay

END SUB
```

```
SUB OpenShutter

' Open shutter and turn on lamp.

        Set (5): ' Open shutter and turn on lamp.

END SUB

SUB PCutility

' Run PCUTILQB.BAS or PCUTILQB.EXE

        CLOSE #1
        CLOSE #2
        CHAIN "C:\qb45\thesis\PCUTILQB"

END SUB

SUB PositionCursor (row, col)

     CLS
     LOCATE 8, 20, 1, 6, 7: 'Moves cursor to middle of screen
     row = 8
     col = 20

END SUB

SUB PrintMessageToOperator

' Print message to operator

        DO
          CALL PositionCursor(row, col)
          BEEP
          INPUT ; "Are drum and trolleys positioned [Y/N] "; Y$
          LOCATE row, col + 1: 'Position cursor to original location
        LOOP UNTIL Y$ = "Y" OR Y$ = "y"

END SUB

SUB RaiseDrumTable

' Raise Drum Table to top postion.

        Set (7): ' Raise drum to top position
        Clr (7): ' Clear relay setting
        Pause (50): ' Wait for drum table to reach top
        Pause (27)

END SUB

SUB ReadOutSpectrum
```

```
' Read spectrum from MCA and store in data file

' Send MCA command to get number of channels in memory range.

      PRINT #2, ESC$; "MEM "; MEM; "#"

' Read memory range channel value sent from MCA

      INPUT #1, MSIZE%

' Send command to MCA for parallel transfer.

      PRINT #2, ESC$; "DOU1; 2; 0; ;"; MEM; "#"

' Call subroutine to open parallel driver for input from the MCA.

      CALLS OPENBI

' The first 2 values of the array are the number of channels
' of data being transferred and the start channel.  Read these
' values in using the INBIN subroutine.

      i = 1
      IAR(i) = 2
      CALLS INBIN(IAR(i))

' The parallel driver will transfer data in 256 channel groups.
' Set up a loop to read in all of the data in 256 channel groups.

      FOR i = 1 TO IAR(1) - 255 STEP 256
      IAR(i + 2) = 256
      CALLS INBIN(IAR(i + 2))
      NEXT i

' Close parallel driver.

      CALLS CLOSEB

' Send signal to stop READ OUT, otherwise MCA might hang.

      PRINT #2, ESC$; "ABT #"

' Write data to data file.

      OPEN DATAFile FOR OUTPUT AS #3

' LOTUS can handle a maximum of 2048 records.  Store the data in
' groups of 2048 channels.  This file will hold a 8192 channel spectrum.

      FOR i = 3 TO 2050
      PRINT #3, USING "#########"; IAR(i); IAR(i + 2048); IAR(i + 4096); IAR(i + 6144)
      NEXT i
```

```
' Close data file.

        CLOSE #3

END SUB

SUB SendSpectrum

' Send a spectrum to MCA

        OPEN DATAFile FOR INPUT AS #3

' Read data into array.

        FOR i = 3 TO 2050
        INPUT #3, IAR(i), IAR(i + 2048), IAR(i + 4096), IAR(i + 6144)
        NEXT i

' Send MCA command to get number of channels in memory range.

        PRINT #2, ESC$; "MEM "; MEM; "#"

' Read memory range channel value sent from MCA

        INPUT #1, MSIZE%

' Send command to MCA for parallel readin.

        PRINT #2, ESC$; "DIN1; 2;"; MEM; "#"

' Write spectrum into MCA

' Open the open parallel driver for output from the CPU to MCA.

        CALLS OPENBO

' The first 2 values of the array are the number of channels
' of data being transferred and the start channel.  Set the first
' array value to the # of channels being transferred.  The second
' value should be 0.

        IAR(1) = MSIZE%
        IAR(2) = 0
        CALL OUTBIN(VARSEG(IAR(1)), VARPTR(IAR(1)))

' Close parallel driver.

        CALLS CLOSEB

' Send signal to stop READ IN, otherwise MCA might hang.

        PRINT #2, ESC$; "ABT #"
```

```
' Close file.

       CLOSE #3

END SUB

SUB SetDisplayColors

       CLS
       COLOR 14, 1

END SUB

SUB SetPIO24

' Initialize PIO-24 mode

' Sets PB port output and PCG-3 input

       SetCRconfig

' Clear all seven PIO-24 relays

       FOR i = 1 TO 7
       Clr (i)
       NEXT i

END SUB

SUB StartDrumProcedure

' This subroutine lowers the drum by half segments and then
' includes the appropriate MCA command file

       FOR Segment = 1 TO NumberSegments

         IF Segment = 1 THEN
           CALL HalfSegmentDrop
         ELSEIF Segment >= 2 OR Segment <= NumberSegments THEN
           CALL HalfSegmentDrop
           CALL HalfSegmentDrop
         END IF

       CALL PositionCursor(row, col)
       PRINT TAB(col); "Scanning Segment:"; Segment;
       LOCATE row, col + 19

       IF ICOM = 4 THEN
         REM $INCLUDE: 'C:\QB45\THESIS\CALIB.BAS'
       ELSEIF ICOM = 5 THEN
         REM $INCLUDE: 'C:\QB45\THESIS\INIT.BAS'
```

```
        ELSEIF ICOM = 6 THEN
          REM $INCLUDE: 'C:\QB45\THESIS\ASSAY.BAS'
        END IF

        CALL MessagePositioningDrumTable

        NEXT Segment

END SUB

SUB TimerDelay

' 2 second time-delay routine

        x = TIMER
        DO
          x1 = TIMER
        LOOP UNTIL ABS(x - x1) >= 2

END SUB
```

Appendix H: Listing of CALIB.BAS

```
' Program called CALIB.BAS, this is an INCLUDE file.
' Save as a text file only!
' Subroutines and Functions are not allowed!
' Used in conjunction with DrumTAB.BAS (menu program)
' CALIB issues command to the CANBERRA MCA to perform a calibration procedure
'
'*****************************************************************************

' The information below contains the preset time and ROI data
' Edit the information below for any particular application
' No other changes need be made elsewhere in the program

ROIdata:  DATA 1,"CS 137",950,973, 2,"PU 238",1104,1127, 3,"Allen",2,11,4,"Da-
ve",1200,1210,0,0,0,0
PresetTime% = 4: 'Total time in seconds for data collection of one segment

'*****************************************************************************

' Clear Data

        PRINT #2, ESC$; "CLD "; MEM; "#"
        Pause (1)

' Set Preset Time

        PRINT #2, ESC$; "PST "; PresetTime%; "; "; MEM; "#"
        Pause (1)

' Start Collecting Data

        PRINT #2, ESC$; "SCO "; MEM; "#"
        Pause (1)

' Wait until data collection is done

        x = TIMER
        DO
          xx = TIMER
        LOOP UNTIL ABS(xx - x) > PresetTime%

OPEN DrumID$ + ".dat" FOR APPEND AS #5

        IF Segment = 1 THEN
          offset = 20
          PRINT #5, "*****************************************************************************"
          PRINT #5, " "
          PRINT #5, "CALIB measurement"
          PRINT #5, "Drum ID#: "; TAB(30); DrumID$
          PRINT #5, "Date: "; TAB(30); DATE$
```

```
          PRINT #5, "Time: "; TAB(30); TIME$
          PRINT #5, "Number of Segments: "; TAB(30); NumberSegments
          PRINT #5, "Preset Time (sec): "; TAB(30); PresetTime%
          PRINT #5, " "
          PRINT #5, "ROI Data"
          RESTORE ROIdata
          READ ROI#, ROI$, ROIstart, ROIend
          NumberOfROI% = 0
          DO WHILE ROI# <> 0
          PRINT #5, ROI$
          PRINT #5, "Start Channel: "; TAB(30); ROIstart
          PRINT #5, "End Channel: "; TAB(30); ROIend
          READ ROI#, ROI$, ROIstart, ROIend
          NumberOfROI% = NumberOfROI% + 1
          LOOP
          PRINT #5, " "
          PRINT #5, TAB(offset); "Counts"; TAB(offset + 10); "Counts Bkg"; TAB(offset +
25); "Counts Net"; TAB(offset + 40); "Segment"
          PRINT #5, " "
          DIM TotalCounts(1 TO NumberOfROI%)
          DIM TotalCountsBKG(1 TO NumberOfROI%)
          DIM TotalCountsNet(1 TO NumberOfROI%)
        END IF

RESTORE ROIdata

FOR ROIindex = 1 TO NumberOfROI%

' Get ROI Data

        READ ROI#, ROI$, ROIstart, ROIend

        PRINT #2, ESC$; "RCD "; ROIstart; ";"; ROIend; ";"; MEM; "#"
        Pause (1)

        INPUT #1, StartChannel$
        INPUT #1, LastChannel$

        Start = VAL(StartChannel$)
        Last = VAL(LastChannel$)
        Channels = ABS(Last - Start) + 1

        DIM ChannelData(1 TO (Channels + 1)) AS STRING

        FOR j = 1 TO (Channels + 1)
          INPUT #1, ChannelData(j)
        NEXT j
```

```
        Counts = 0
        FOR j = 1 TO Channels
          Counts = Counts + VAL(ChannelData(j))
        NEXT j
        CountsBKG = Channels * (VAL(ChannelData(1)) + VAL(ChannelData(Channels))) / 2

        PRINT #5, TAB(1); ROI$;
        PRINT #5, USING "######.##"; TAB(offset - 4); Counts; TAB(offset + 10); CountsBKG;
TAB(offset + 25); Counts - CountsBKG;
        PRINT #5, TAB(offset + 45); Segment

        Pause (1)

        TotalCounts(ROIindex) = Counts + TotalCounts(ROIindex)
        TotalCountsBKG(ROIindex) = CountsBKG + TotalCountsBKG(ROIindex)
        TotalCountsNet(ROIindex) = (Counts - CountsBKG) + TotalCountsNet(ROIindex)

        ERASE ChannelData

NEXT ROIindex

PRINT #5, " "

IF Segment = NumberSegments THEN

        PRINT #5, " "
        PRINT #5, "Total Counts"
        RESTORE ROIdata
        FOR index = 1 TO NumberOfROI%
          READ ROI#, ROI$, ROIstart, ROIend
          PRINT #5, ROI$;
          PRINT #5, USING "######.##"; TAB(offset - 4); TotalCounts(index); TAB(offset +
10); TotalCountsBKG(index); TAB(offset + 25); TotalCountsNet(index)
        NEXT index
        PRINT #5, " "
        PRINT #5, "************************************************************************"
        PRINT #5, " "
        PRINT #5, " "
        CLOSE #5
        OPEN DrumID$ + ".dat" FOR INPUT AS #5
        DO UNTIL EOF(5)
          LINE INPUT #5, LineBuffer$
          LPRINT LineBuffer$
        LOOP
        ERASE TotalCounts
        ERASE TotalCountsBKG
        ERASE TotalCountsNet

END IF

RESTORE ROIdata: ' return start of next read to first data statement
CLOSE #5
```

Appendix I: Listing of MakeDrum

```
#               Make file called MakeDrum
#               To run, at DOS prompt type: MAKE MAKEDRUM

TableSYS.OBJ:   TableSYS.ASM
                MASM TableSYS.ASM

DrumSYS.OBJ:    DrumSYS.ASM
                MASM DrumSYS.ASM

PCUTILQB.OBJ:   PCUTILQB.BAS
                C:\qb45\bc PCUTILQB.BAS /zi /v /x /w /o

PCUTILQB.EXE:   PCUTILQB.OBJ
                LINK PCUTILQB.OBJ,PCUTILQB,,c:\qb45\bcom45.lib /co

DrumTAB.LIB:    DrumSYS.OBJ TableSYS.OBJ
                LIB DrumTAB.LIB + DrumSYS.OBJ + TableSYS OBJ

DrumTAB.OBJ:    DrumTAB.BAS CALIB.BAS ASSAY.BAS INIT.BAS DrumTAB.LIB
                c:\qb45\bc DrumTAB,DrumTAB,DrumTab /zi

DrumTAB.EXE:    DrumTAB.OBJ
                LINK DrumTAB,,,c:\qb45\bcom45.lib + c:\qb45\thesis\DrumTAB.LIB /co
```

Appendix J: Electrical Mapping of PIO-24 and STC

Sense Line	PIO-24	ERA-01	STC
1	PA 1	pin 36	J2-1
2	PA 2	pin 35	J2-2
3	PA 3	pin 34	J2-3
4	PA 4	pin 33	J2-4
5	PA 5	pin 32	J2-5
Control Line	**PIO-24**	**ERA-01**	**STC**
1	PB 1	PBR 1 NO	J2-9
2	PB 2	PBR 2 NO	J2-10
3	PB 3	PBR 3 NO	J2-11
4	PB 4	PBR 4 NO	J2-12
5	PB 5	PBR 5 NO	J2-13
6	PB 6	PBR 6 NO	J2-14
7	PB 7	PBR 7 NO	J2-15
Gnd	NA	PBR 1-7 CO	J2-25

NO: Normally Open Contacts
CO: Common Contacts
PB: PB Port
PA: PA Port
PBR: PB Port Relay

Appendix K: Drum Measurement Run Using DrumTAB

****

CALIB measurement
Drum ID#: 33416
Date: 11-27-1990
Time: 09:30:07
Number of Segments: 8
Preset Time (sec): 3

ROI Data
CS 137
Start Channel: 950
End Channel: 973
PU 238
Start Channel: 1104
End Channel: 1127

	Counts	Counts Bkg	Counts Net	Segment
CS 137	152.00	0.00	152.00	1
PU 238	1.00	0.00	1.00	1
CS 137	122.00	0.00	122.00	2
PU 238	1.00	0.00	1.00	2
CS 137	134.00	0.00	134.00	3
PU 238	1.00	0.00	1.00	3
CS 137	141.00	12.00	129.00	4
PU 238	3.00	12.00	-9.00	4
CS 137	139.00	0.00	139.00	5
PU 238	0.00	0.00	0.00	5
CS 137	129.00	0.00	129.00	6
PU 238	0.00	0.00	0.00	6
CS 137	131.00	0.00	131.00	7
PU 238	0.00	0.00	0.00	7
CS 137	119.00	0.00	119.00	8
PU 238	1.00	0.00	1.00	8

Total Counts

CS 137	1067.00	12.00	1055.00
PU 238	7.00	12.00	-5.00

****

Appendix L: Derivation of Correction Factors

Wall and matrix material correction factors are derived below for incorporation into CAL-IB.BAS at a later time.

Activity

After selecting a signature gamma of interest, the activity, $A(t)$, of a sample can be calculated by

$$A(t) = \lambda N(t) \left(\frac{dis}{\sec}\right) \qquad (L.1)$$

where λ = decay constant and $N(t)$ = number of atoms present at time t. Additionally, a specific decay rate, $A_{sa}(t)$ or specific activity, can be defined by

$$N \left(\frac{atoms}{gram}\right) = Av/AtomicMass \qquad (L.2)$$

$$\lambda = \left[\frac{ln2}{T_{1/2}}\right] \qquad (L.3)$$

$$A_{sa}(t) = \frac{ln2}{T_{1/2}} \times \left[\frac{Av}{AtomicMass}\right] \left(\frac{dis}{g-\sec}\right) \qquad (L.4)$$

where

Av = Avogadros Number,
$T_{1/2}$ = half-life of the material.

Since the decay of most radioisotopes is not always followed by a sole gamma emission, the specific decay rate is more appropriately defined as

$$A_s(t) = f \lambda N \left(\frac{\gamma}{g-\sec}\right) \qquad (L.5)$$

where f (γ/dis) is the fraction that decays by the signature gamma of interest.

To convert activity to count rate and vice versa, count rate, CR, is given by

$$CR = \epsilon_{abs} A_s m \left(\frac{counts}{\sec}\right) \qquad (L.6)$$

where m is the mass of unknown sample in grams, and ϵ_{abs} is the absolute detector efficiency in counts/gamma ray.

Now one can measure the number of counts deposited in a detector and relate this to specific activity:

$$CR = \frac{C\lambda}{(1 - e^{-\lambda t_c})} \qquad (L.7)$$

where t_c is the counting time, and C is the net number of counts under the signature gamma full-energy peak (excluding background). Furthermore, CR needs to be corrected for dead time losses, if appreciable.

In the particular case for ^{238}Pu, where the half-life is fairly long, i.e., if $\lambda t_c \ll 1$, $e^{-\lambda t_c} \approx 1 - \lambda t_c$ by a Taylor expansion. Thus,

$$C \approx \frac{\epsilon_{abs} A_s m \lambda t_c}{\lambda} \qquad (L.8)$$

or

$$A_s m \approx \frac{C}{\epsilon_{abs} t_c} \ (Bq) \qquad (L.9)$$

At first it appears the right-hand side of Equation (L.9) contains only one unknown; however, ϵ_{abs} could easily be obtained by performing an assay with a calibration drum of known activity. Yet there is another factor that has been neglected, that is attenuation of the source in the drum itself. Using Equation (L.9) alone would not account for this, and it is derived next.

Transmission

A drum consists of a matrix material, unknown source or sources, and the outer walls. The parts of the matrix material and drum walls are of dissimilar substances so they attenuate the source signature gamma rays differently. This attenuation causes the observed count rate (Equation (L.7)) to differ from the ideal case in which neither the matrix material or drum walls are present, i.e., the count rate noticed in absence of attenuating materials. As a result data collected from the MCA will predict a wrong activity value, however, using an additional "transmission" source in conjunction with the unknown drum source will correct for most deviations.

First a distinction must be made between the different sources; hence the unknown source is designated the assay source. The assay source is not confined to a single area, and cannot be treated as a point source. Instead, it is uniformly distributed (almost always) throughout the entire drum among the matrix material. About the only complication present here, besides drum attenuation, is a single scan of one drum segment does not provide sufficient data to analyze the entire drum inventory. Scanning multiple segments (otherwise known as mapping) and rotating the drum normally averages out any inhomogeneities caused by source geometry[7].

Furthermore, the walls are usually considered separately, that is, not part of the matrix material. Thus drum attenuation is caused by two factors. One is wall attenuation, and the second is matrix material attenuation. Both add to the overall attenuation of the source radiation.

The following derivation assumes wall and matrix material attenuation are the same, i.e., both are indistinguishable from each other and are lumped together under the general category of drum attenuation. Drum attenuation can be determined by using a transmission source with known activity emitting a gamma ray close in energy to that of the assay source signature gamma. Here, attenuation of the transmission source is defined by

$$T_t = \frac{T_{att}}{T_o} = e^{-\left(\frac{\mu}{\rho}\right)_t \rho_d x_t} \tag{L.1.1}$$

where

T_t = transmission source attenuation,
T_{att} = attenuated intensity of transmission source,
T_o = unattenuated intensity of transmission source,
$\left(\frac{\mu}{\rho}\right)_t$ = mass attenuation coefficient of drum at transmission source gamma-ray energy,
ρ_d = density of drum,
x_t = transmission source pathlength (or distance through the drum).

Similarly, attenuation of the assay source is

$$T_a = e^{-\left(\frac{\mu}{\rho}\right)_a \rho_d x_a} \tag{L.1.2}$$

where

T_a = assay source attenuation,
$\left(\frac{\mu}{\rho}\right)_a$ = mass attenuation coefficient of drum at assay source signature gamma energy,
ρ_d = density of drum,
x_a = assay source pathlength (or distance through the drum).

It also follows that

$$A_{meas} = A_{corr} \times T_a \tag{L.1.3}$$

such that A_{meas} is the measured assay source intensity from Equation (L.9), and A_{corr} is the intensity of the assay source, corrected for drum attenuation.

In a simple example where both transmission and assay sources experience the same attenuation and their difference in pathlengths is accounted for by some geometry factor, F_{geo}, for assay and transmission intensities close together[8]

$$A_{corr} = \frac{A_{meas}}{T_a} = A_{meas}\frac{F_{geo}}{T_t} = A_{meas} \times \frac{T_o}{T_{att}} \times F_{geo} \tag{L.1.4}$$

Also, for assay and transmission sources far apart in energy

$$T_a = e^{-\left(\frac{\mu}{\rho}\right)_a \rho_a x_a} = T_t^{\left\{\frac{\left(\frac{\mu}{\rho}\right)_a}{\left(\frac{\mu}{\rho}\right)_t r_{geo}}\right\}} \qquad (L.1.5)$$

Thus

$$A_{corr} = A_{meas} \times \left[\frac{T_o}{T_{att}}\right]^{\frac{\left(\frac{\mu}{\rho}\right)_a}{\left(\frac{\mu}{\rho}\right)_t r_{geo}}} \qquad (L.1.6)$$

As has been noted, wall and matrix material attenuation were considered inseparable. Next, individual corrections for wall and matrix material are addressed. Expanding Equation (L.1.3) for the two contributions yields

$$A_{meas} = A_{corr} \times e^{-\left(\frac{\mu}{\rho}\right)_{wall} \rho_{wall} x_{wall}} \times e^{-\left(\frac{\mu}{\rho}\right)_{mn} \rho_{mn} x_{mn}} \qquad (L.1.7)$$

$$A_{corr} = A_{meas} \times F_{wall} \times F_{mm} \qquad (L.1.8)$$

where F_{wall} is the correction factor for attenuation due to the drum walls, and F_{mm} is the correction factor for attenuation due to the matrix material inside the drum.

F ast the unattenuated activity/intensity of the assay source can be calculated by applyi.., Equations (L.9) and (L.1.8). In addition, since mass or number density is proportional to activity, the assay source quantity inside the drum can be determined.

Derivation of Wall Attenuation Factor, F_{wall}. Drum wall attenuation is accounted for by

$$F_{wall} = \frac{1}{\sqrt{T_{wall}}} = \left(\frac{T_o}{T_{empty}}\right)^{\frac{k}{2}} \qquad (L.1.1.1)$$

where

$$T_{wall} = \text{drum wall transmission} = \left(\frac{T_{empty}}{T_o}\right)^k,$$

and where T_{empty} is the transmission intensity thru an empty drum, T_o is the unattenuated transmission intensity, and k is the ratio of mass attenuation coefficients (of drum wall) at assay and transmission energies; $k = 1$ if the energies are very close together.

The square root of T_{wall} is used since the transmission gamma rays pass through 2 walls of the drum whereas assay gamma rays pass through only 1 wall thickness. By neglecting air attenuation inside an empty drum, the transmission source attenuation is

$$T_t^{wall} = e^{-\left(\frac{\mu}{\rho}\right)_t^{wall} \rho_{wall} 2x_{wall}} \qquad (L.1.1.2)$$

where

T_t^{wall} = transmission source attenuation in drum wall,
$\left(\frac{\mu}{\rho}\right)_t^{wall}$ = mass attenuation coefficient of drum wall at transmission source energy,
ρ_{wall} = drum wall density,
x_{wall} = thickness of one drum wall.

Then the assay source attenuation in the drum wall, T_a^{wall}, is given by

$$T_a^{wall} = e^{-\left(\frac{\mu}{\rho}\right)_a^{wall} \rho_{wall} x_{wall}} = T_t^{\frac{\left(\frac{\mu}{\rho}\right)_a^{wall}}{2\left(\frac{\mu}{\rho}\right)_t^{wall}}} \qquad (L.1.1.3)$$

where $\left(\frac{\mu}{\rho}\right)_a^{wall}$ is the mass attenuation coefficient of the drum wall at the assay source energy. But from Equation (L.1.1)

$$T_t^{wall} = \frac{T_{att}}{T_o} \Rightarrow T_t^{wall} = \frac{T_{empty}}{T_o} \qquad (L.1.1.4)$$

hence

$$T_a^{wall} = \frac{T_{empty}}{T_o}^{\left\{\frac{\left(\frac{\mu}{\rho}\right)_a^{wall}}{2\cdot\left(\frac{\mu}{\rho}\right)_t^{wall}}\right\}} \qquad (L.1.1.5)$$

And

$$k = \frac{\left(\frac{\mu}{\rho}\right)_a^{wall}}{\left(\frac{\mu}{\rho}\right)_t^{wall}} ,$$

$$\therefore\ T_a^{wall} = \left(\frac{T_{empty}}{T_o}\right)^{\frac{k}{2}} = \sqrt{T_{wall}} \qquad (L.1.1.6)$$

Lastly, from Equation (L.1.3)

$$A_{corr} = \frac{A_{meas}}{T_a} = A_{meas} \times \frac{1}{\sqrt{T_{wall}}} \Rightarrow F_{wall} = \frac{1}{\sqrt{T_{wall}}} \qquad (L.1.1.7)$$

Derivation of Matrix Material Attenuation Factor, F_{mm}. The transmission source attenuation (assuming a homogeneous matrix material) is

$$T_t^{mm} = e^{-\left(\frac{\mu}{\rho}\right)_t^{mm}\rho_{mm}x_{mm}} \qquad (L.1.2.1)$$

where

T_t^{mm} = transmission source attenuation in matrix material,
$\left(\frac{\mu}{\rho}\right)_t^{mm}$ = mass attenuation coefficient of matrix material at transmission source energy,
ρ_{mm} = matrix material density,
x_{mm} = thickness through matrix material.

Then the assay source attenuation in the matrix material, T_a^{mm}, is given by

$$T_a^{mm} = e^{-\left(\frac{\mu}{\rho}\right)_a^{mm}\rho_{mm}x_{mm}\beta} =$$

$$e^{\frac{-\left(\frac{\mu}{\rho}\right)_t^{mm}\rho_{mm}x_{mm}\left(\frac{\mu}{\rho}\right)_a^{mm}\beta}{\left(\frac{\mu}{\rho}\right)_t^{mm}}} = T_t^{mm\left\{\frac{\left(\frac{\mu}{\rho}\right)_a^{mm}\beta}{\left(\frac{\mu}{\rho}\right)_t^{mm}}\right\}} = (T_t^{mm})^{k^{mm}\beta} \qquad (L.1.2.2)$$

where

$\left(\frac{\mu}{\rho}\right)_a^{mm}$ = mass attenuation coefficient of matrix material at assay source energy,
β = geometric correction factor[9] (.823 for cylinders),
k^{mm} = ratio of mass attenuation coefficients in the matrix material at assay and transmission source energies.

But from Equation (L.1.1)

$$T_t = \frac{T_{att}}{T_o} = e^{-\left(\frac{\mu}{\rho}\right)_t^{wall}\rho_{wall}x_{wall}2}\, e^{-\left(\frac{\mu}{\rho}\right)_t^{mm}\rho_{mm}x_{mm}}$$

$$\Rightarrow T_t^{mm} = \frac{T_{non\text{-}empty}}{T_o T_t^{wall}} \qquad (L.1.2.3)$$

where $T_{non\text{-}empty}$ is the measured transmission intensity through a full or non-empty drum.

Thus from Equation (L.1.2.2)

$$T_a^{mm} = \left(\frac{T_{non\text{-}empty}}{T_o T_t^{wall}}\right)^{k^{mm}\beta} \qquad (L.1.2.4)$$

And finally, from Equation (L.1.3)

$$A_{corr} = \frac{A_{meas}}{T_a} \Rightarrow A_{meas} \times \frac{1}{T_a^{mm}} \Rightarrow F_{mm} = \frac{1}{T_a^{mm}} \qquad (L.1.2.5)$$

Bibliography

1. Campbell, A.R. Et Al. "Assay of *Pu*^{238} Contaminated Waste In Drums'" *MD-21825, Issue 1*, 3, EG&G Mounds Laboratory, Dayton OH, Apr 1985.

2. East, Larry V. "Subroutines Callable from a PDP-11 BASIC for Control of a Multichannel Pulse-Height Analyzer," *LA-5772-MS*, Los Alamos Scientific Laboratory, November 1974.

3. *CANBERRA PC Toolkit Software Manual*, Canberra Industries, August, 1988.

4. Microsoft. *Microsoft Macro Assembler 5.1 Programmer's Guide*. Redmond, Wash. 1986, 1987.

5. Microsoft. *Microsoft Macro Assembler 5.1 Mixed-Language Programming Guide*. Redmond, Washington. 1987.

6. Canberra Segmented Gamma Scanner, Model 2220B (55 Gallon Drums), Operating Manual Version 3, April, 1978.

7. Martin, E.R.,D.F. Jones, and J.L. Parker. "Gamma-ray Measurements with the Segmented Gamma Scan," *LA-7059-M*, Los Alamos Scientific Laboratory, December 1977.

8. Model 2220 Segmented Gamma Scanner, *Technical Reference Manual*, Canberra Industries, Inc., Nuclear Systems Division, Meriden, CT, Jan 1979 ,p 2-3.

9. *Canberra Model 2220 Segmented Gamma Scanner Technical Reference Manual*, Canberra Industries, January, 1979.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1991	Master's Thesis

4. TITLE AND SUBTITLE HARDWARE UPGRADE OF A SEGMENTED DRUM ASSAY SYSTEM	5. FUNDING NUMBERS
6. AUTHOR(S) Claude A. Irvine, Captain, USAF	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB OH 45433-6583	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GNE/ENP/91M-3
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This paper describes the conversion of a DEC PDP-11/05 computer system, previously used in Canberra's Model 2220B segmented gamma scanner, with an IBM PC. Two tasks necessary for completion of the project involved reestablishing communications with a Canberra Series 35+ multi-channel analyzer and a scan table controller. An additional serial/parallel card was installed in the PC to reinstate communications with the multi-channel analyzer. For computer control of scan table operations a digital input/output card was used along with an external electromechanical relay board; when implemented together this hardware setup replaces functions that were normally processed through a motion control interface card housed within the DEC. Software consisted of Canberra's PC Toolkit while newer programs were written in Microsoft's QuickBASIC 4.5 and Macro Assembler 5.1. Five codes were written--two of these are device drivers written in assembly language and the other three are menu and control programs written in QuickBASIC. The modification enables simplified programmer enhancements.

14. SUBJECT TERMS Gamma Ray Spectroscopy, Radiation Measuring Equipment Drum Assay Equipment, DEC PDP-11/05, Radioactive Wastes, Canberra Segmented Gamma Scanner	15. NUMBER OF PAGES 92 16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL